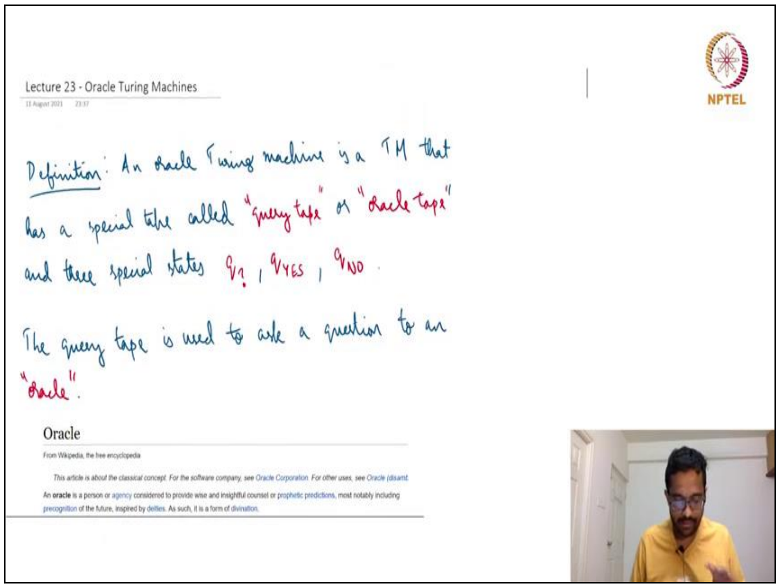
Computational Complexity

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

Lecture -23
Oracle Turing Machines

(Refer Slide Time: 00:17)



Lecture 23 - Oracle Turing Machines

NPTEL

Definition: An oracle Turing machine is a TM that has a special tape called "query tape" or "oracle tape" and three special states $q_?$, q_{YES}, q_{NO}.

The query tape is used to ask a question to an "oracle".

Oracle

From Wikipedia, the free encyclopedia

This article is about the classical concept. For the software company, see Oracle Corporation. For other uses, see Oracle (disamt

An **oracle** is a person or agency considered to provide wise and insightful counsel or prophetic predictions, most notably including precognition of the future, inspired by deities. As such, it is a form of divination.

Hello and welcome to lecture 23 of the course computational complexity. In this lecture we will see oracle turing machines. So, we have seen reductions and oracle which are many-to-one reductions; many one reductions. Oracle turing machines are another way to view reductions using what is called an oracle. So, if you if you Google for the word oracle obviously there is this this software company called Oracle Corporation.

But if you for the english meaning of the word oracle. So, this oracles were people who were in like some in ancient times or mythologies or whatever people who used to be maybe in a maybe in the courts of kings or maybe otherwise who used to provide kind of advice supposedly from the gods or from the future or some somewhat prophecy type of things. I have pasted a snippet of the wikipedia entry for the word oracle.

So, they are people. So, oracles were people who were consulted in order to get like advice. So this is keeping this in mind let us try to understand the word what is what are oracle turing machines. So, oracle turing machines are turing machines which have access to an oracle what are oracles. So, oracles could be; so, we will we will first define oracle as a specific problem or specific language. So, it could be for instance I think in this lecture mostly we will use the SAT oracle.

So you know we know that SAT is an NP complete language. So, a SAT oracle if you write down a set instance if you write down a Boolean formula the oracle will immediately tell you whether this formula is satisfiable or not. So, this is what the oracle does. So, you get the yes or no answer immediately and the correct answer. So, it is not just any yes or no answer. So, an oracle turing machine has the functionality or has a has a way to ask the oracle queries and get responses and can use these responses in its computation.

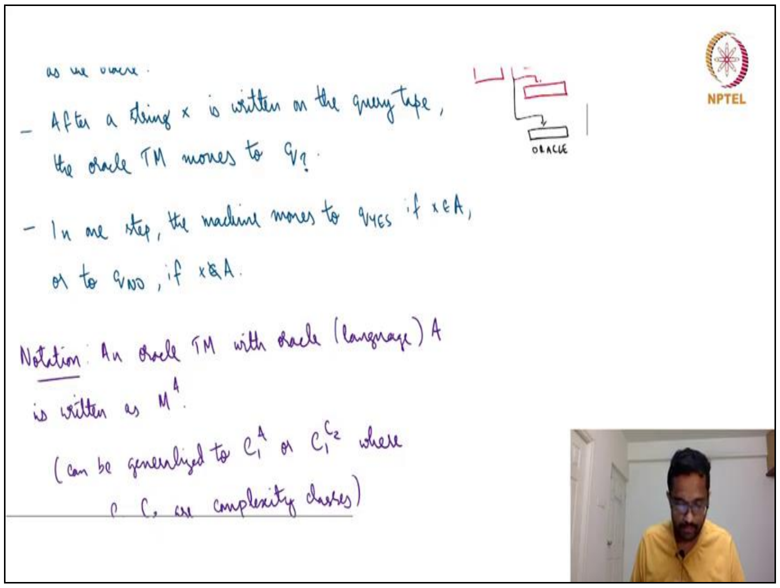
So, it can make the SAT query many times if it has a SAT oracle it may not be a SAT oracle it could be any oracle. So, it could be a three colourable oracle. So, whenever you give it a graph it immediately tells you whether the graph is three colourable or not. So, the oracle is well it is an oracle. So, it does not take any time you give it, it immediately gives you the answer back. So, if you give the question it immediately gives the answer back it does not spend it take any time it is just one step.

So, how is it like how is it formalized in this notion. So, it is a turing machine and it has a special tape called the oracle tape or the query tape. And on the query tape you the turing machine will write queries. So, queries or it need not write in one step it could be constructed in the sequence of many steps. But and so, if it is a SAT oracle it will write a certain instance. So, when the query when the SAT instances or the query is ready at some point it will go to the it will go to a special state.

The special state being the q query this is the first one is a quick queue query denoted by q with a question mark in subscript some text books may say query. And then the next step the oracle gives the response how does the oracle give the response it will look at the query tape or the oracle tape. So, it does not need time to read or anything it is just one step and then it will immediately move the turing machine to the yes state or the no state.

So, this again this yes state and the no state are also special state. So, all of these three are special states q query, q yes and q you know. So, if it is a yes instance it will move to q yes if it is a no instance it will move to q no. So, this is what it does.

(Refer Slide Time: 04:52)



as the oracle.

- After a string x is written on the query tape, the oracle TM moves to $q_?$.
- In one step, the machine moves to q_{YES} if $x \in A$, or to q_{NO}, if $x \notin A$.

Notation: An oracle TM with oracle (language) A is written as M^A.

(can be generalized to C_1^A or $C_1^{C_2}$ where C_1, C_2 are complexity classes)

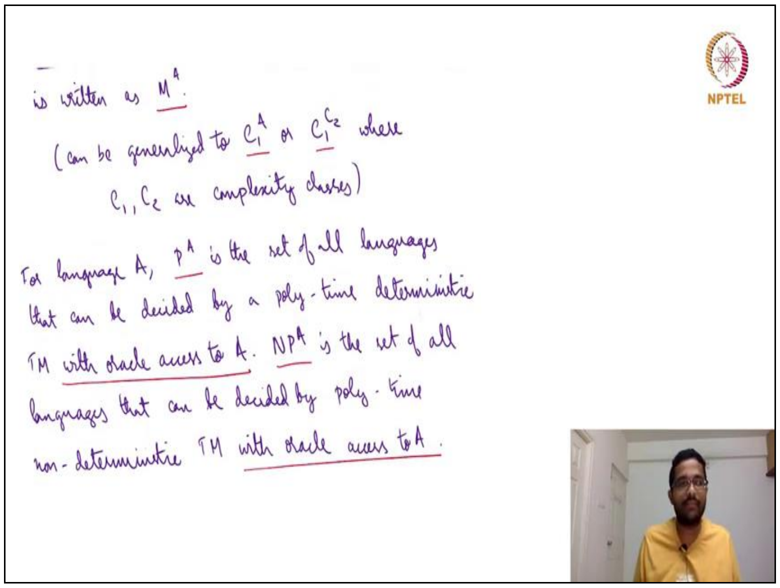
ORACLE
NPTEL

And an oracle turing machine is a turing machine with these properties and this is the oracle could be any language like as I said it could be SAT it could be three colourable it could be subset some it could be like two colourable anything. So, depending on how powerful the language that corresponds to the oracle is you will find different we will find will find different results. So, how does it function? I have already described but I will just go through it again the oracle is the; the oracle corresponds to some language.

So, here we are denoting that language as A. So, it has access to the A oracle the language A oracle. So in what I described the oracles possible are like it could be like I said SAT or three colourable in it any language can be used as an oracle and at some point this the string is written on the query tape or the oracle tape maybe I will just try to. So, there are turing machine tapes let us see this is the in the work tapes.

Let us say these two are the work tapes and let us see the black one is the oracle tape. So, this is the oracle tape the other ones are work tapes or the query tape. So, it will keep filling up the oracle tape and then it will move to a state the oracle state q query or the query state. And then depending on what is there on the oracle tape it will move to q yes or queue no. So, if it is a yes sense it will move to q yes. This is what basically it will the oracle will check whether the string is a part of A or not a part of A.

(Refer Slide Time: 06:55)



is written as M^A.

(can be generalized to C_1^A or $C_1^{C_2}$ where C_1, C_2 are complexity classes)

For language A, P^A is the set of all languages that can be decided by a poly-time deterministic TM with oracle access to A. NP^A is the set of all languages that can be decided by poly-time non-deterministic TM with oracle access to A.

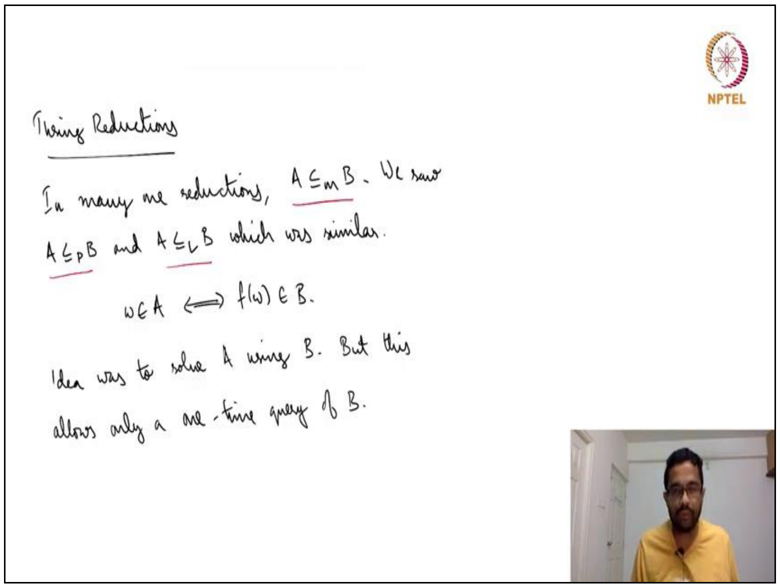
This is what happens and such an oracle is called or such a turing machine is called if a turing machine m is there with access to an oracle A it is denoted by m with superscript A. So, it has m and it has access to the oracle A and we will see later we will see generalizations generalizations of this where we will see complexity classes with access to an oracle. So, we will define what they are we will soon see. And we can even define again this we will not quite do but we will we can we can do that if you see textbooks you may see this notation where complexity classes have complexity classes as oracles.

So, where both the; subscript as well as the superscript are both complexity classes. So, let me just define the first one C 1 with access to a means. So, it is a complexity class. So, first we will use P superscript A to define it. So, basically you have a polynomial time deterministic polynomial time turing machine. So, what is P? P is a collection of languages decidable by deterministic polynomial time turing machines.

But now P superscript a means it is the class of languages decided by deterministic polynomial type turing machines it is exact exactly what it was for P but with access to an a oracle but with access to an a oracle. So, this red underline part was not there for P. So, that is the access to the oracle. NP superscript A is again the same thing it is; so, what is NP? NP is a class of languages decidable but non-deterministic polynomial time turing machines NP superscript A is the class of languages that are decidable by non-deterministic polynomial time turing machines with access to an oracle A.

So, again the red underlined part at the end of the sentence is what changes. So, this is what a complexity class superscript game means it is the same machines that form part of the complexity class but with access to an oracle A, c 1 superscript c 2 will not define now just to avoid making too many definitions this is the notation.

(Refer Slide Time: 09:37)



Turing Reductions

In many one reductions, $A \leq_m B$. We saw $A \leq_p B$ and $A \leq_L B$ which was similar.

$$w \in A \iff f(w) \in B.$$

Idea was to solve A using B. But this allows only a one-time query of B.

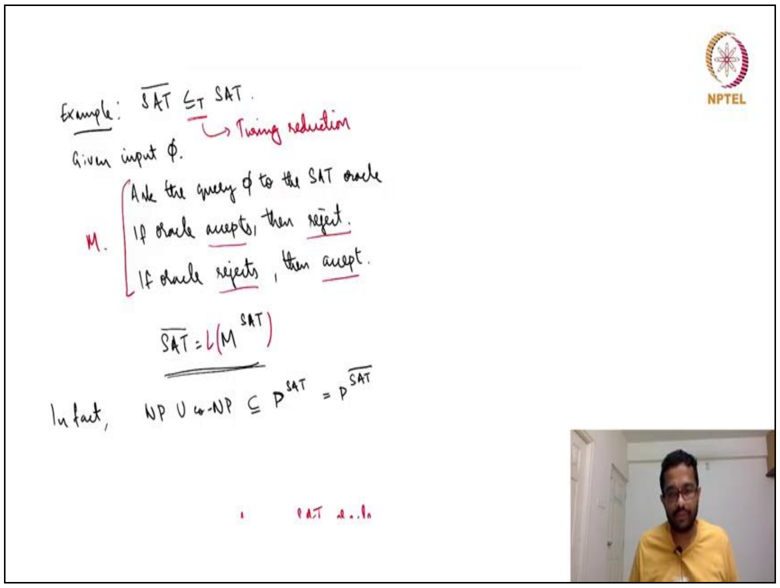
So, what is the reason for making this definition or what is the motivation behind this. So, we have seen many one reductions which is denoted A less than or equal to subscript m B. So, we although we did not define it in this course you must have seen it in the theory of computation course that in your undergrad. But the same principle applies for the polynomial time and log space reductions both of which we saw in this course.

So, what was the principle there. So, the principle was that f sorry W is in A if and only if f of w is in B. So, basically we are transforming an instance of A into an instance of B. So, you are transforming A instance w to a B instance f of w and then depending and then using that to solve A. So, basically using converting a instance to B instance and then using an algorithm of B to solve A, correct. This is what we did in the two in the many one reductions or polynomial time or log space reduction.

So, many one reductions is just the same thing except that there is no restriction on the time or space used by the machine but polynomial time and log space are specific cases of many one reductions. So, tuning reductions which is what we are going to define now or we is some. So, there is one issue or there is one specific limitation with the mini one reductions or log space or polynomial time reduction.

So, basically you convert the problem instance and then use the decider for B to decide A. So, this can be viewed as asking the decider of B just once. So, you are asking you just convert it and then ask the decider for B is it in B or not is the converted instance in B or not. So, basically we are getting only one query to ask the B decider. So, why not ask the B decider many times. Do we get any advantage out of that? Do we get any benefit by allowing us the greater possibilities of asking B many times.

(Refer Slide Time: 11:58)



Example: $\overline{SAT} \leq_T SAT$.

$\hookrightarrow$ Turing reduction

Given input ϕ.

M. Ask the query ϕ to the SAT oracle
If oracle accepts, then reject.
If oracle rejects, then accept.

$\overline{SAT} = L(M^{SAT})$

In fact, $NP \cup co\text{-}NP \subseteq P^{SAT} = P^{\overline{SAT}}$

So that is what turing reduction does. So, turing deduction is basically an access to an oracle of B. So, when you have an access to an oracle you could ask queries to the oracle multiple times not only once. So, the question is; question here with many one reductions was sorry why cannot we query B multiple times and that is what is addressed by the turing reduction. So, basically we say A turing reduces to B if A is decided by a turing machine m with access to an oracle B or B oracle.

So, we will see one simple example. So, one is which is very, very simple and maybe you might find you might feel it is silly but even this simple or silly example illustrates something illustrates the power of turing reduction over many one reductions. So, the simple example is that SAT complement is reducible to SAT using turing reduction. So, subscript less than or equal to subscript T indicates turing reduction.

SAT complement is reducible to SAT. So, what does it mean it means that there is a machine with access to a SAT oracle that can decide set complement. So, given a set complement instance so, let us say given a formula phi it is in SAT complement if it is not satisfiable or in other words if it is not satisfiable it is in SAT complement which means it is a yes and sense of SAT complement. So, which means it is a no instance of SAT.

So, basically all that we do is we ask phi to the SAT oracle if the SAT oracle says that it is a yes instance you make you answer no and this hat oracle says it is a no instance you answer yes. So, which is what is written here if the SAT oracle accepts your reject and if it rejects you accept. So basically you flip the output of the SAT oracle. So, you start with an input you just make this you just write down the input in the query tape ask the SAT oracle one query and flip the output.

So, what we have shown here is if this is a machine m that does this we have shown that SAT complement is decided by may be that small is decided by is a language of this machine m with access to SAT oracle. And if you notice all that m is doing is just writing down the input in the query tape which can be done in linear time and then flipping the output. So, this is certainly linear time.

So, m is a polynomial time machine a deterministic polynomial time machine. So, I can as well say that so we can actually say that SAT complement is in sorry that complement is in P superscript SAT it is in P with a with access to a SAT oracle because m is a polynomial time deterministic polynomial time machine. In fact it is not very difficult to see that even SAT maybe I will write it above it is not very difficult to see that even SAT is sorry SAT is in deterministic polynomial time with SAT oracle with access to SAT oracle.

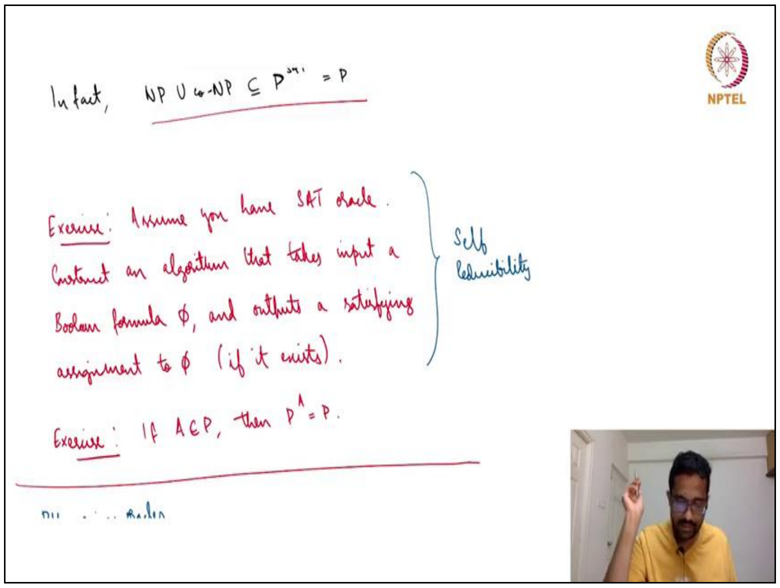
Because it is it is trivial because you want to solve SAT and you have access to the set oracle then all the machine has to do is just copy the query copy into the query tape and make the query and then accept or reject accordingly the same thing without flipping. And in fact now SAT is a NP complete language SAT complement is a co-NP complete language and we have seen that using turing reduction you can reduce a co-NP complete language to an NP complete language which was not possible if we just used polynomial time reductions as we had learnt.

So, you see that turing reduction is actually more powerful even though this particular example was extremely simple you see that even the simple example illustrates this fact. So, SAT is an NP complete language and SAT complement is a co-NP complete language. So, in fact both of them are contained in P superscript SAT which means that the entire class NP is contained in P superscripts at P with access to SAT oracle.

Because SAT is contained in P superscript squared because you could take any NP instance you can reduce it to a SAT instance and then ask the and SAT is in P superscript SAT. And co-NP likewise you can reduce it to S SAT complement instance and even that has been shown to be in P superscript SAT. So, both NP and co-NP are in P with access to a SAT oracle. And notice that P with access to a SAT compliment oracle is the same thing.

Because if you want if you get yes no answer a SAT compliment will tell you whether it is not satisfiable or satisfiable it is the same thing that SAT oracle also tells you just that it answers the flipped output. So, you take the answer of the SAT oracle and flip it you get the SAT complement. So, P superscripts SAT which is equal to P superscripts SAT complement ah. So, NP and co-NP are both contained in that.

(Refer Slide Time: 18:34)



In fact, $NP \cup co\text{-}NP \subseteq P^{SAT} = P$

Exercise: Assume you have SAT oracle. Construct an algorithm that takes input a Boolean formula ϕ, and outputs a satisfying assignment to ϕ (if it exists).

Self Reducibility

Exercise: If $A \in P$, then $P^A = P$.

One maybe two exercises the first exercise is to kind of see how or to for you to see how powerful is this idea of making multiple queries to a certain oracle can be. Suppose you have a SAT oracle which tells you yes or no given a SAT instance whether it is satisfiable or not. Now try please construct an algorithm which given a Boolean formula if it is not satisfiable then you just output it is not satisfiable.

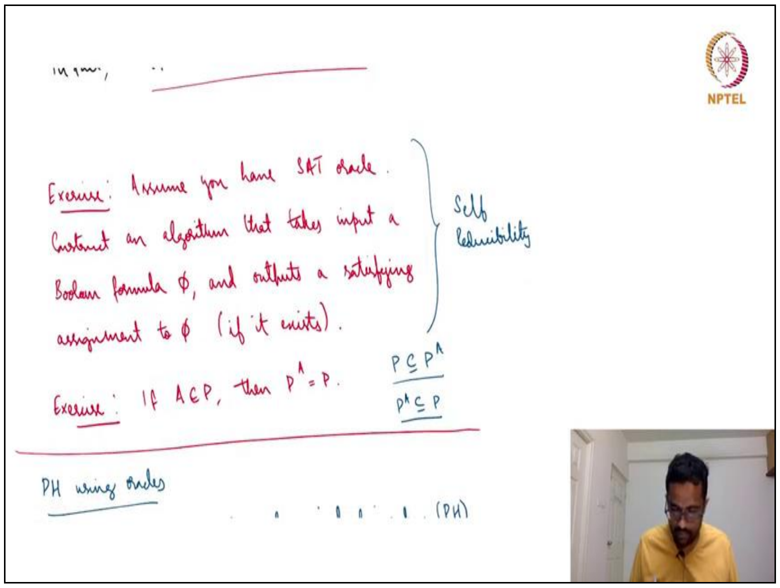
If it is satisfiable then actually the algorithm should provide and should output an assignment a satisfying assignment for the formula. So, if given phi it should tell if i is satisfiable it should give it should tell a satisfying assignment for the formula phie. So, try to construct this using a SAT oracle and you will nee and what I am telling you now is that you will need to make multiple calls, calls to the SAT oracle, multiple queries to the SAT oracle.

But still you can find it. And the idea used in this is it is not that difficult to find but if you think hard you can find it. And the idea used in this will be useful in for a theorem that we will see later in the course. So, this the idea used here maybe I will just tell the name of the idea it is called self reducibility. But anyway you do not need to know what self reducibility is etcetera to answer this exercise just try to see.

So, maybe what the first step is let us say you given a formula phi you just give phi to the SAT oracle. So, SAT oracle will say whether it is satisfiable or not. If it is not satisfiable you are done because there is nothing to be done. If it is not satisfiable there is no satisfying assignment. If it is not satisfiable you are done. If it is satisfiable then you have the task of producing a satisfying assignment.

Think how you will go about this. So, remember the SAT oracle can take any formula any instance of status satisfiability as an input. So, what instances can we give think about this.

(Refer Slide Time: 21:16)



Exercise: Assume you have SAT oracle. Construct an algorithm that takes input a Boolean formula ϕ, and outputs a satisfying assignment to ϕ (if it exists).

Self Reducibility

Exercise: If $A \in P$, then $P^A = P$.

$P \subseteq P^A$

$P^A \subseteq P$

PH using Oracles

(PH)

Next exercise is to show that if A is in P then the class P with access to an A oracle is P itself. So, we saw that P with NP is contained in P with access to a SAT oracle and P and co-NP are both contained but however that was for SAT if A is in P then P with access to an A oracle is actually P itself. So, obviously P is contained the this direction is easy P is contained in P with access to an A oracle.

So, usually just again this is one of these things I will just say whenever you have to show that two sets are equal two classes here in this case are equal one of the standard ways to go about doing this is to show that the left hand side is a subset of the hand side and the hand side is a subset of the left hand side. So in this case P is obviously contained in P with access to an A oracle because P is well P itself.

So, even without accessing even without accessing the A oracle we can say P is. So, air equal only adds more power but the other direction P with access to A oracle we have to show that it is contained in P this will be again this is not very difficult to see because A itself is in P or A is also a polynomial time. A is also decidable in polynomial time. So instead of asking the oracle this machine itself can run the algorithm for A.

So, that is why the second the second the other direction P with access to A oracle is contained in P that is how that comes about that is a slightly; that that will require a bit of explanation but the other one is even easier and yeah that is what I want to say in this mini lecture. Just define oracle turing machine which is a special type of turing machine which allows us to ask an oracle some queries.

And I defined m superscript a as a machine m with access to an oracle A and P with superscript A is the complexity class machines in complexity class B with access to an oracle A. Likewise NP superscript A we define turing reductions as instead of reducing it to that language you can have oracle access to that language. So, turing reduction is A to be released to B there is an oracle turing machine m says that A is the language of m with access to oracle B.

And finally some simple inferences using for these complexity classes and the exercises and that is all I have for lecture 23, thank you.