E0 224 Computational Complexity Theory **Indian Institute of Science, Bangalore**
Fall 2014 **Department of Computer Science and Automation**

Lecture 8: Sep 4, 2014

Lecturer: Chandan Saha <chandan@csa.iisc.ernet.in> *Scribe: Pawan Kumar*

8.1 Recap of the last Lecture

1. **Two features/properties of diagonalization based proofs**
   (a) Every Turing Machine can be represented by binary strings.
   (b) There is a universal Turing Machine that can simulate any other TM say M without much overhead with running time within a log factor of the running time of M.
2. **Remark**
   The P vs NP problem can not be resolved using just the two properties mentioned above.The reason is that "Any proof that uses only the above two properties must necessarily relativize. However we will show in today's lecture that any P vs NP problem/proof must necessarily involve a 'Non-relativizing' technique."

8.2 Oracle Machines and Limits of Diagonalization

The notion of relativization uses "Oracle Turing Machine".We will first define what is an Oracle TM both deterministic as well as Non-deterministic.

8.2.1 Oracle Turing Machine

Definition 8.1.

1. *An Oracle Turing Machine is a TM that has a special read-write tape called M's oracle tape and three special states* $q_{query}, q_{yes}, q_{no}$.
2. *To execute M, we specify the input as usual;and a language* $O \subseteq \{0,1\}^*$ *that is used as an oracle for M.*
3. *While performing its computation, if M enters the state* q_{query} *then M checks whether the contents of the oracle tape* $w \in O$*. If* $w \in O$*, M moves to the state* q_{yes}*, else it moves to* q_{no}.
4. *Regardless of the choice of O,a query like* $w \in O$ *counts for a single step of M.*
5. $M^O(x)$ *denotes the output of the oracle TM M on input* $x \in \{0,1\}^*$ *with* $O \subseteq \{0,1\}^*$ *as the oracle.*

We say that M^O is deterministic Oracle Turing Machine if M is deterministic,otherwise if M is Non-deterministic we say M^O is a Non-detrerministic Oracle Turing Machine.

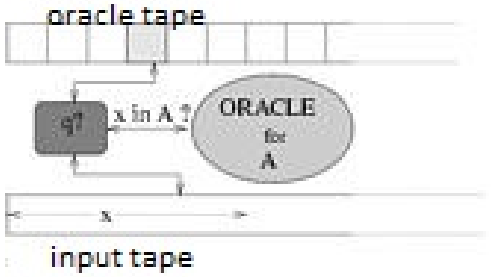

Figure 8.1: a diagrametic view of OTM

Definition 8.2. *A language $L \subseteq \{0,1\}^*$ is in the complexity class P^O (where O is an arbitrarily fixed language) if there exist a deterministic polytime Oracle Turing Machine that decides L.*

Definition 8.3. *A language $L \subseteq \{0,1\}^*$ is in the complexity class NP^O (where O is an arbitrarily fixed language) if there exist a Non-deterministic polytime Oracle Turing Machine that decides L.*

Question : Is $\mathbf{NP} = \mathbf{P^{SAT}}$?

It is conjectured that $NP \subsetneq P^{SAT}$

$\mathbf{NP} \subsetneq \mathbf{P^{SAT}}$: Let L be a language in NP.By the Cook-Levin theorem there's a poly-time reduction mapping a string x to a boolean formula ϕ_x s.t. $x \in L \iff \phi_x \in SAT$. On input x we can compute ϕ_x and query the SAT oracle of the Oracle Turing Machine in P^{SAT}.

$\mathbf{P^{SAT}} \subseteq \mathbf{NP}$: This statement is conjectured to be FALSE as it is also conjectured $NP \neq Co - NP$ and it can be observed that Co-NP is also contained in P^{SAT}.

Claim : $\overline{SAT} \in P^{SAT}$ where $\overline{SAT}$ denotes the language of unsatisfiable formulae i.e. $\overline{SAT} = \{\phi : \phi$ is unsatisfiable $\}$

Proof. We have oracle access to SAT which returns true iff the formula given to it is satisfiable. A deterministic poly-time Oracle Turing Machine can query its oracle if $\phi \in SAT$ and then give the opposite answer. □

Form the above claim,we conjecture that $\mathbf{NP} \neq \mathbf{P^{SAT}}$.

Claim: If $O \in P$ then $P^O = P$

Proof.

1. It trivially follows $P \subseteq P^O$.
2. If $O \in P$,then replace each oracle call with a deterministic poly-time computation of O.
3. Number of oracle calls by TM M can be atmost polynomial and product of two polynomial is another polynomial. Hence, $P^O \subseteq P$.

□

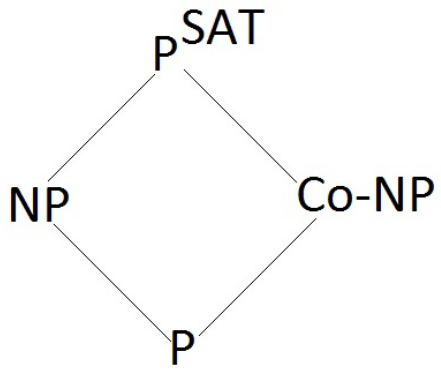

Figure 8.2: Complexity Class Hierarchy

Question : Does there exist an oracle A s.t. $P^A = NP^A$?

CANDIDATE A :

EXPCOM= $\{< M, x, 1^n >$: M is a deterministic TM M that accepts x in 2^n steps $\}$.

Claim : EXPCOM is EXP-Complete language under poly-time reduction.

1) EXPCOM is in EXP i.e. EXPCOM $\in$ EXP.

2) Let L$\in EXP$ i.e. there exists a TM M s.t. M accepts x in exponential time.

Reduction $L \leq_P EXPCOM$: Let $x \in L$ and p(n) is the amount of time taken by TM M to accept x. A poly-time reduction function f which reduces x to $< M, x, 1^{p(n)} >$, s.t. $x \in L \iff < M, x, 1^{p(n)} > \in EXPCOM$. Both side implications are self explanatory.

Claim : $P^{EXPCOM} = NP^{EXPCOM}$

Proof.

1. $EXP \subseteq P^{EXPCOM}$,this holds trivially.Since EXPCOM is EXP-Complete language,there's a polynomial time reduction f s.t.$x \in L \iff f(x) \in EXPCOM$. Once we compute $f(x)$ form x we query oracle EXPCOM of the Oracle Turing Machine in P^{EXPCOM}.
2. $P^{EXPCOM} \subseteq NP^{EXPCOM}$,this also holds trivially.
3. $NP^{EXPCOM} \subseteq EXP$,if M is a Non-deterministic poly-time oracle TM,we can simulate its execution with a EXPCOM oracle in exponential time:Such time suffices both to enumerate all of M's nonderministic choices and to answer the EXPCOM oracle queries.

□

prop1 : Oracle O is arbitrarily fixed.Every Oracle TM with access to O can be represented by finite length binary strings.

prop2 : There is a universal TM with access to O that can simulate any other turing machine M^O on input x without much overhead.

Definition 8.4. *We say a proof relativizes,if the same proof works w.r.t. any arbitrary oracle O.*

Remarks :

1. In principle this fixed language(corresponds to oracle O) may be undecidable but in this course we will assume it to be decidable.
   - In case the oracle is a language it returns either 0 or 1 depending on whether query string is in language or not.
   - The oracle may return a general boolean string instead of a single bit assume 0 or 1.
2. Any proof that strictly uses the two properties of diagonalization mentioned above relativizes.This means any result about standard TMs (TMs without access to oracle) that uses only diagonalization(meaning the two properties) also holds for Turing Machines with access to any arbitrary oracle.
3. It may happen that the result/proof about an Oracle Turing Machine,say M, does not hold when the oracle is detatched from M. The reason could be the use of some specific properties of the oracle in the proof.
4. The power of the oracle effects the proofs involving P vs NP problems.The result of the problems may differ when we use powerful oracle as compare to a weak oracle.

Theorem 8.5. *There exists languages(oracles) A,B such that* $P^A = NP^A and P^B \neq NP^B$.

Proof. We have already proved $P^A = NP^A$ if A is EXPCOM. The same result also holds when A is PSPACE-Complete.(This may be a good exercise.)

Now we have to find a language say B such that $P^B \neq NP^B$,but why this theorem is important for us(what we are going to achieve?).The following discussion tries to answer this question.

1. By using the theorem stated above one can show diagonalization techniques solely based on the two properties will not resolve the P vs NP question.
2. If $P \neq NP$ is provable using the two properties,then even if assistance of oracle is given then $P^O \neq NP^O$ should hold for all oracles O,but there exist an oracle A such that $P^A = NP^A$.
3. If $P = NP$ is provable using the two properties,then even if assistance of oracle is given then $P^O = NP^O$ should hold for all oracles O ,but there exist an oracle B such that $P^B \neq NP^B$.
4. The two points stated above clearly shows that if the stated theorem is TRUE then proof solely using the two properties will not resolve the P vs NP question.

Proof Sketch ($P^B \neq NP^B$) : For an arbitrarily language B let U_B be the unary language $U_B = \{1^n$:there exist a string of length n in B $\}$.

□

Claim : $U_B \in NP^B$,This claim holds trivially, Given a string 1^n, we guess a string x of length n and ask the oracle 'is x in B?'.This computation is performed by the oracle in one step.Hence $U_B \in NP^B$.Since, we did not restrict B to have any particular property,this result holds for any B.

Now we construct B in such a manner that $U_B \notin P^B$. We will include strings in B in stages ,Construction of B is an iterative process which uses the strings which are currently present in B.

Stage i : just before stage i we have decided membership of finitely many strings in B.

1. Initially B is empty and each stage determines the status of a finite number of strings.
2. For every i,let M_i^B be the oracle TM represented by binary expansion of i.
3. Let n_i be the number strictly larger than the length of a longest string whose "fate" has been decided.
4. Simulate M_i^B on input 1^{n_i} for $\frac{2^{n_i}}{10}$ steps.
5. Whenever M_i^B queries the oracle B with strings whose 'fate' has already been decided,we answer consistently,
6. If M_i^B queries a string 'y' whose 'fate' has not been decided yet,then we decide the 'fate' of that string by not including it in B.(hence oracle answers no to such queries).
7. If M_i^B does not halt or outputs 0 after $\frac{2^{n_i}}{10}$ steps then include a string of length n_i in B that has not been queried by M_i^B.(Surely there exist one such string because there are 2^{n_i} strings are there and M_i^B might have queried atmost $\frac{2^{n_i}}{10}$ strings of length n_i). One can pick string in a lexicographically the smallest and discard the rest.
8. If M_i^B Outputs 1,then don't include any string of length n_i in B.

Claim : $U_B \notin P^B$

We can see that at any stage i answer of M_i^B on 1^{n_i} is incorrect.If M_i^B outputs 1 then according to (8) there is no string of length n_i in B,which implies $1^{n_i} \notin U_B$.Similarly if M_i^B doesn't halt or outputs 0 then according to (7) there's a string of length n_i in B,which implies $1^{n_i} \in U_B$.Hence in either case the answer of M_i^B is incorrect.Since every polynomial $p(n)$ is smaller than $\frac{2^n}{10}$ for large n,and every Turing machine M is represented by infinitely many strings,our construction will ensure that M does not decide U_B in polynomial time [M1].

References

[M1] S. ARORA and B. BARAK "Computational Complexity: A Mordern Approach," *Cambridge University Press*, 2009