CSE531: Computational Complexity Theory January 18, 2012

Lecture 4 More on NP and the Limits of Diagonalization

Lecturer: Anup Rao

1 Boolean Formulas

A *boolean formula* is an expression of the form

$$(x_1 \wedge \neg x_2) \vee (x_7 \wedge \neg(x_6 \vee \neg x_2)).$$

Formally: it is a circuit where the only allowed gates are $\vee, \wedge, \neg$, and every gate has fan-out at most 1. Input gates are allowed to repeat. As usual, size of the gates is number of gates, and the fan-in is allowed to be at most 2. The formula is said to be in conjunctive normal form (CNF) if it is an AND of OR's. Similarly, it is said to be in disjunctive normal form if it is an OR of ANDS. For example

$$(x_1 \vee \neg x_2) \wedge (\neg x_7 \vee x_9 \vee \neg x_1)$$

is a CNF.

We have the following lemma:

Lemma 1. *Every function $f : \{0,1\}^\ell \to \{0,1\}$ can be computed by a CNF (resp. DNF) of size $\ell 2^\ell$.*

Proof For each input z such that $f(z) = 0$, we add the literal x_i to the clause if $z_i = 0$ and $\neg z_i$ otherwise. So for example, if $f(0,1,0) = 0$, we add the clause $(x_1 \vee \neg x_2 \vee x_3)$. Then note that each clause is 0 on exactly one input, and all inputs that make f 0 make some clause 0. The resulting formula is of size $\ell 2^\ell$. The case of DNF's is symmetric. ■

We define $\mathsf{SAT} : \{0,1\}^* \to \{0,1\}$ to be the function that takes as input a boolean formula F, and outputs 1 if and only if there is a an x such that $F(x) = 1$.

Theorem 2. SAT *is* **NP***-complete.*

Proof $\mathsf{SAT} \in \mathbf{NP}$ is easy enough to check. The witness is a satisfying assignment to the formula.

Since we have already shown that $\mathsf{CKT-SAT}$ is **NP**-hard, it will be enough to show that $\mathsf{CKT-SAT} \leq_P \mathsf{SAT}$.

Given a circuit, we shall output a CNF formula that is satisfiable if and only if the circuit accepts some input. Introduce a new variable y_g for each internal gate g of the circuit. If the internal gate g has inputs h, q, let F_g be the CNF formula on variables y_g, y_h, y_q that is 1 if and only if $y_g = g(y_q, y_h)$. By Lemma 1, this formula is of constant size. If the output gate is v, the final formula is

$$y_v \wedge \bigwedge_g F_g,$$

which is satisfied if and only if the circuit has a satisfying assignment. ■

A 3-CNF formula is a CNF where every clause has at most 3 variables. For example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \neg x_1) \wedge \cdots$.

$\mathsf{3SAT} : \{0,1\}^* \to \{0,1\}$ is the function that takes as input 3-CNF and outputs 1 if and only if the formula is satisfiable. Next we show that even this function is **NP**-complete

Theorem 3. 3SAT *is* **NP***-complete.*

Proof Clearly, 3SAT $\in$ **NP** .

Any clause $(a \vee b \vee c \vee d)$ can be replaced with $(a \vee b \vee z) \wedge (a \vee b \vee \neg z)$, and the clause is satisfiable if and only the resulting two clauses are satisfiable. In this way, we can convert any CNF formula into a 3-CNF, such that the final formula is satisfiable if and only if the original formula is satisfiable. ■

Remark 4. *Is the same true for* 2SAT*? We do not know. There are polynomial time algorithms for* 2SAT*, so if you found a reduction to* 2SAT*, you would prove* $\mathbf{P} = \mathbf{NP}$*.*

There are many natural **NP**-complete problems. For example:

- Independent set. Given a graph G and a number k, compute whether or not the graph has an independent set of size k.

  To show that this problem is **NP**-hard, we can reduce from 3SAT. Replace each 3SAT clause with a triangle of vertices, one for each literal, all connected. Connect literals across triangles if and only if they contradict each other. If the formula is satisfiable, then there is an independent set of size equal to the number of triangles, since each clause must contain a satisfied literal, which gives a vertex from each triangle. Conversely, if there is an independent set of size m, then consider the assignment that satisfies each literal of the set. This is assignment is well defined by the construction of the graph, and it must satisfy the formula.

- Hamiltonian path. Given a directed graph G, is there a path that visits every vertex exactly once?

2 Nondeterministic Machines

The original definition of **NP** was by considering Turing machines that are allowed to make non-deterministic choices: namely after each step, the machine is allowed to make a guess about which state to transition to in the next step. The machine computes 1 if there is a single accepting computational path, and 0 otherwise.

We can define $\mathsf{NTIME}(t(n)$ in the same way as $\mathsf{DTIME}(t(n))$, it is the set of functions computable by non-deterministic machines in time $O(t(n))$, and then you can check that $\mathbf{NP} = \bigcup_c \mathsf{NTIME}(n^c)$. Just as for deterministic time, there is a non-deterministic time hierarchy theorem:

Theorem 5. *If* r, t *are time-constructible functions satisfying* $r(n+1) = o(t(n))$*, then*

$$\mathsf{NTIME}(r(n)) \subsetneq \mathsf{NTIME}(t(n)).$$

3 The problem with diagonalization as a technique for separating P and NP.

Definition 6 (Oracle Machines)**.** *Given a function* $O : \{0,1\}^* \to \{0,1\}$*, an* oracle-machine *is a Turing Machine that is allowed to use a special oracle tape to make queries to* O*. Each query takes unit time.*

We can define $\mathbf{P}^O$, $\mathbf{NP}^O$ as functions computable in poly time (resp nondeterministic poly time) with oracle access to O.

Then we have the following theorem:

Theorem 7. *There exists an oracle A such that $\mathbf{P}^A = \mathbf{NP}^A$, and an oracle B such that $\mathbf{P}^B \neq \mathbf{NP}^B$.*

The theorem gives a hint about one of the ways in which it will be hard to determine whether or not $\mathbf{P} = \mathbf{NP}$. Any such proof must not work in the *relativized* worlds where access to A, B is permitted. **Proof** Let A be the function that on input α, x outputs 1 if and only if $M_\alpha(x)$ outputs 1 in $2^{|x|}$ steps. Then $\mathbf{P}^A = \mathbf{EXP}$, since every exponential time computation can be simulated with access to A, and every query to A can be simulated in exponential time. Also $\mathbf{NP}^A = \mathbf{EXP}$, since in exponential time we can simulate all queries to A and simulate all nondeterministic choices.

The second part is more interesting. We shall define an oracle $B : \{0,1\}^* \to \{0,1\}$ and a function $f \in \mathbf{NP}^B$ such that $f \notin \mathbf{P}^B$. Once we have defined B, we define f as follows:

$$f(x) = \begin{cases} 1 & \text{if there exists } y \text{ such that } |y| = |x| \text{ and } B(y) = 1, \\ 0 & \text{else.} \end{cases}$$

$f \in \mathbf{NP}^B$, since a machine can guess the y of the same length as x, and make a single query to verify that $B(y) = 1$.

To define B, we shall use diagonalization. Our goal is to make sure that the $i'th$ machine fails to compute the correct value of $f(x)$ in time $2^{n/10}$, for some n. To do this we define the value of B gradually.

We define the value of B in phases. After each phase, we shall have defined the value of B on a finite set of strings. In Phase i, let t be so large that the value of B is not yet defined on each string of length t. Then run the i'th machine $M_i(1^t)$ for $2^{t/10}$ steps. Each time M_i queries a string of B whose value has not yet been defined, return 0 (and define the value of B on that string to be 0. If M_i halts with value 1, then set B to be 0 on all strings of length t. If M_i halts with value 0, then pick a string y of length t that M_i did not query (if such a string exists), and set its value to $B(y) = 1$.

Set the value of B on strings that are not defined by the above process to be 0.

Suppose for the sake of contradiction that $f \in \mathbf{P}^B$. Then consider the machine M that computes f. Since each machine is represented an infinite number of times, and M runs in polynomial time, there must be some i for which the number t is so large that $2^t/10$ exceeds the running time of M on inputs of length t. Then in phase i, M cannot possibly query all input strings of length t, and so M will compute the wrong value for f. ■

4 NP versus the set of NP-complete problems

Theorem 8. *If $\mathbf{P} \neq \mathbf{NP}$, there is a function $f \in \mathbf{NP} - \mathbf{P}$ that is not $\mathbf{NP}$-complete.*