Lecture 11

September 28, 2005

1 Review: The Complexity Class $\mathcal{NP}$

Last time we formalized the notion of "hard to find, but easy to check". A language $L \subseteq \Sigma^*$ is in the class NP if there is a polynomial P and a PTIME algorithm A such that for every $x \in \Sigma^*$: $x \in L \Leftrightarrow A(x, y) = 1$ for some $y \in \Sigma^{\leq P(|x|)}$. Intuitively, verification that $x \in L$ (signified by the two-argument algorithm A returning 1) takes only a polynomial number of steps. The string y is called the 'witness' for x, since it "witnesses" the fact that $x \in L$. Thus being in NP means both that (1) a short witness exists and (2) that it can be verified in polynomial time.

One can equivalently view NP in terms of *non-determinism* (the name 'NP' stands for Nondeterministic Polynomial time); that is, we can imagine an algorithm that just guesses an answer and then checks it in polynomial time. We shall adopt the first view. (For a good exposition of the non-deterministic view see the text by Hopcroft and Ullman.) It is important to note that either condition (1), (2) or both can fail. For instance, the problem of determining if white will win at a game of chess is not in NP since neither (1) nor (2) hold. Hilbert's 10th problem[1] satisfies condition (2) but not condition (1).

2 Reducibility and NPC

We also formalized a notion of reducibility, intuitively the process of reducing one problem to another using an "efficient" (i.e. polynomial) algorithm.

We defined a relation $\leq_p$ for languages $L_1, L_2 \subseteq \Sigma^*$: $L_1 \leq_p L_2 \Leftrightarrow$ there is a PTIME-computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in L_1$ iff $f(x) \in L_2$. We say that L_1 is reducible to L_2 since L_1 is "no harder" than L_2 in the sense that if L_2 is solvable then so is L_1. Moreover, if $L_1 \leq_p L_2$ and $L_2 \in PTIME$ then $L_1 \in PTIME$. This is easy to see since a polynomial algorithm A' that decides the language L_2 can be used to decide L_1 by using the function f to transform $x, x \in L_1$ into $f(x)$, which the algorithm A' can then decide.

[1] In its original form the problem is: "Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers."

Recall also that we defined a language $L \subseteq \Sigma^*$ to be **NP-complete** if $L \in \mathrm{NP}$ and if $L' \leq_p L$ for every $L' \in \mathrm{NP}$. A language L satisfying the second condition is said to be **NP-hard**. **NPC** is a short name for the class of NP-complete languages.

If it is the case that $L_1 \leq_p L_2$ and $L_2 \leq_p L_1$, we say that $L_1 \equiv_p L_2$. The relation $\equiv_p$ defines a set of equivalence classes within NP, ordered by $\leq_p$. We might naturally ask whether there are 'top' and 'bottom' elements in this pre-order.

Theorem 1. *There is a maximal equivalence class of $\equiv_p$ in NP, namely NP-Complete, or NPC.*

Intuitively, we say that NPC problems are the hardest problems in NP. Similarly, we can also say that PTIME is at the bottom because of polynomial reducibility. But what is in between? Moreover we might ask whether $\mathrm{P} = \mathrm{NP}$ or $\mathrm{P} \neq \mathrm{NP}$. We don't know! The concept of NP-completeness is important because it is at the crux of the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ question:

Theorem 2 (Ladner). *If $P \neq NP \Rightarrow (NP - NPC) - P \neq \emptyset$. Equivalently, $P \neq NP \Rightarrow NP - (NPC \cup P) \neq \emptyset$.*

Theorem 3. $NPC = P \;\Rightarrow\; P = NP$.

3 Co-NP and CoNP-Completeness

Recall that a formula φ is satisfiable iff $(\neg\varphi)$ is not valid, and that φ is valid iff $(\neg\varphi)$ is not satisfiable. Thus the two decision problems SAT and VALID are related. It is reasonable to ask whether $SAT \equiv_p VALID$?

Suppose that this is indeed the case. Then there is some function $f : Expr \rightarrow Expr$ such that φ is SAT iff $f(\varphi)$ is VALID. But we have a problem because while we have a witness for SAT (namely, a truth assignment), it is not immediately clear what we can use as a witness for validity. (In fact, it can be shown that a short witness for validity doesn't exist). However, notice that if we wish to prove that validity doesn't hold, then a short witness exists–namely, a truth assignment which makes the formula as a whole return 0.

It turns out that the class of problems that have short counterexamples is interesting in itself. The next definition formalizes this concept.

Definition 1 (Co-NP). *A problem (language) $L \subseteq \Sigma^*$ is in Co-NP is there is a polynomial P and a PTIME algorithm A, such that for all $x \in \Sigma^*$, we have $x \notin L$ iff there is some $y \in \Sigma^{\leq P(|x|)}$ such that $A(x, y) = 0$.*

Observe that in NP, when given an answer we can 'easily' verify that it is indeed a solution, while for Co-NP, when given something that is *not* an answer, we can 'easily' verify that it is *not* a solution.

Lemma 1. *A problem $L \in Co\text{-}NP$ iff $L = \Sigma^* - L'$ for some $L' \in NP$*

Proof. You will prove this result in assignment 3. □

Like the NP-complete class, **NPC**, we can define the Co-NP-complete class, **Co-NPC**, which represents the hardest problems in Co-NP as follows:

Definition 2 (Co-NPC). *A problem $L \in Co\text{-}NPC$ if*

1. $L \in Co\text{-}NP$
2. $L' \leq_p L$ *for all* $L' \in Co\text{-}NP$

Lemma 2. *A problem $L \in Co\text{-}NPC$ iff $L = \Sigma^* - L'$ for some $L' \in NPC$.*

Proof. Again, you will prove this result in assignment 3. □

How are SAT and VALID related? Recall that SAT has two requirements, namely that we have a well-formed formula and that it is satisfiable, or (WFF $\wedge$ SAT). Taking the negation we get ($\neg$wff $\vee$ $\neg$sat). Thus, we cannot exactly say that SAT is the complement of VALID because VALID contains well-formed formulae and not just expressions. Reformulated, $\Sigma^* - SAT \neq VALID$ because while SAT is all satisfiable formulae, $\Sigma^* - SAT$ includes nonsensical expressions as well. However, we can say that $SAT \in NPC$ and $VALID \in Co\text{-}NPC$.

More generally, how are NP and Co-NP related? We know that both contain as a subset the complexity class PTIME. Is $P = NP \bigcap Co\text{-}NP$? Is it possible that NP = Co-NP?

We know that all problems in P are trivially in NP (discussed earlier). P is *also* contained in Co-NP: suppose some problem $L \in P$. If we run the algorithm to solve the problem, then invert the output, we end up with a solver for $\Sigma^* - L$, which is also in P.

Lemma 3. $P \subseteq NP \bigcap Co\text{-}NP$

Do there exist problems in both NP and Co-NP that are *not* in P? In this regard consider the problem of testing compositness (ie, is a number prime or is it a composite?). It is trivial to show that COMPOSITNESS $\in Co\text{-}NP$. Showing that COMPOSITNESS $\in NP$ is attributed to Pratt, 1976. Thus, COMPOSITNESS $\in NP \bigcap Co\text{-}NP$. However, it was not clear until 2002 whether COMPOSITNESS $\in P$. In 2002, Agarwal et al showed that COMPOSITNESS $\in P$. Whether $P = NP \bigcap Co\text{-}NP$ is an open problem.

Recall that P is *closed* under the *complement* operation. So, if $P = NP$, then also $P = Co\text{-}NP$. Open problem: $NP \stackrel{?}{=} Co\text{-}NP$

Lemma 4. *If $L \in NPC$ and $L \in Co\text{-}NP$, then $NP = Co\text{-}NP$*

Theorem 4 (Ladner). *The full glory of Ladner's theorem can be expressed as following:*

- *If* $P \neq NP \Rightarrow (NP - NPC) - P \neq \emptyset$.
- *If* $P = NP \Rightarrow NP = Co\text{-}NP$.
- *If* $NP \neq Co\text{-}NP \Rightarrow P \neq NP$.

4 Looking ahead

Next time we will show that 3-Color $\leq_p$ SAT. This means that in theory, if we have a SAT solver, we can do 3-Coloring. But how practical is this? *Prima facie* it might seem that using a SAT solver would be inferior to a system designed especially for 3-Coloring. However, because a *lot* of work has been done on SAT-solvers, it turns out that using SAT as a generic problem solving environment is not a bad idea.