Computability and Complexity

Lecture 6, Thursday August 17, 2023

Ari Feiglin

What is the significance of **NP**-complete problems? Well, $\mathbf{P} = \mathbf{NP}$ if and only if there exists an **NP**-complete problem which is also in **P**. Obviously if $\mathbf{P} = \mathbf{NP}$ then every **NP** problem, and in particular every **NP**-complete problem is in **P**. And if there exists an **NP**-complete problem in **P**, then every problem in **NP** can be reduced to this problem in polynomial time and therefore is in **P**.

But if $\mathbf{P} \neq \mathbf{NP}$, then does there exist an **NP** problem which is not **NP**-complete but is also not in **P**? It turns out that there is.

Theorem 6.1 (Ladner's Theorem):
If $\mathbf{P} \neq \mathbf{NP}$, then there exists a probem in **NP** which is neither **NP**-complete nor in **P**.

Proof:

The idea is to take an **NP**-complete problem and remove an infinite number of results, while retaining an infinite number of results. So we define a function of the form

$$S = \{x \mid x \in \mathsf{SAT},\ f(|x|) \text{ is even}\}$$

Our goal is to define a function $f\colon \mathbb{N} \longrightarrow \mathbb{N}$ which has the following properties:

(1) f can be computed in polynomial time.

(2) If $S \in \mathbf{P}$ then f is even except for a finite number of inputs. This would be a contradiction, as then S would be equal to SAT minus a finite number of results. But then S would remain **NP**-complete (we could define a reduction from SAT to S), and thus S is in **P** and **NP**-complete, so $\mathbf{P} = \mathbf{NP}$ in contradiction.

(3) If S is **NP**-complete, then f is odd except for a finite number of inputs. This would imply S is finite, and thus in **P**, meaning $\mathbf{P} = \mathbf{NP}$ in contradiction.

So all that remains is to find such a function f. Let us define the sequence of Turing machines which decide search problems in polynomial time

$$M_1^D, M_2^D, \dots, M_n^D, \dots$$

and let us define the sequence of Turing machines which compute some Karp reduction in polynomial time

$$M_1^K, M_2^K, \dots, M_n^K, \dots$$

Let M_{SAT} be a Turing machine which decides SAT (SAT is decidable in exponential time).

Let us define the Turing machine M_f which accepts as input a number in unary, and we define $f(n) = M_f(1^n)$ (1^n means n represented in unary). We define it like so:

function $M_f(1^n)$
  if $(n = 1)$ **return** 1
  $k \leftarrow M_f(1^{n-1})$
  if (k is even)
    $i \leftarrow \frac{k}{2}$
    for ($z \in \{0,1\}^*$ where $|z| \leq \log(n)$)
      ▷ *The idea now is to see if $M_i^D(z)$ returns the correct answer to the question $z \in S$.*
      This is if and only if $M_{\mathsf{SAT}}(n)$ returns one, and $M_f(1^{|z|})$ returns one (meaning $f(|z|)$ is odd).
      if ($M_i^D(z) = 1$ **and** ($M_{\mathsf{SAT}}(z) = 0$ **or** $M_f(z) = 0$)) **return** $k+1$
      else if ($M_i^D(z) = 0$ **and** $M_{\mathsf{SAT}}(z) = 1$ **and** $M_f(z) = 1$) **return** $k+1$
    end for
    return k
    ▷ *So we return $k+1$ if M_i^D gives the wrong answer, and k if it gives the correct answer.*

else if (k is odd)
  $i \leftarrow \frac{k+1}{2}$
  for ($z \in \{0,1\}^*$ where $|z| \leq \log(n)$)
    ▷ *Now we check if M_i^K fails to be a reduction from* SAT *to S.*
    if ($M_f(z) \neq M_{\mathsf{SAT}}(M_i^K(z))$) **return** $k+1$
  end for
  return k
end if
end function

We must show that M_f runs in polynomial time. Note that at each step, we iterate over inputs z whose length is at most $\log(n)$, and then it runs some Turing machines whose time is exponential in $|z|$. There are at most n such inputs, and since $|z|$ is logarithmic in n, being exponential in $|z|$ is polynomial in n. Thus M_f runs in polynomial time, as required.

Notice that $f(n) \leq f(n+1) \leq f(n)+1$, as if $M_f(1^{n-1})$ returns k, then $M_f(1^n)$ returns either k or $k+1$. In other words $f(n+1) \in \{f(n), f(n)+1\}$.

If $S \in \mathbf{P}$ then there exists a Turing machine M_i^D which decides S. Let $k = 2i$, then if $M_f(1^{n^*}) = k$ then for $n = n^*+1$, we get $M_f(1^{n-1}) = k$ and so M_i^D still decides S and so $M_f(1^n)$ will return k. Inductively we see that for every $n^* \geq n$, $f(n) = k$. Now suppose that $M_f(1^n)$ is never k, then there is some other stable point. If this stable point is even, then f is even for all but a finite number of inputs, as required. Otherwise we have a stable odd point, and this means we have a Turing machine M_i^K which forms a reduction from SAT to S, which would mean S is $\mathbf{NP}$-complete and so $\mathbf{P} = \mathbf{NP}$ in contradiction.

And if S is $\mathbf{NP}$-complete, then there exists a Turing machine M_i^K which computes a Karp reduction from SAT to S. Let $k = 2i-1$, then similar to above there exists an n^* such that for every $n \geq n^*$, $f(n) = k$ and so f is odd except for a finite number of times. ■

Proposition 6.2:

The class of $\mathbf{NP}$ is closed under Karp reductions. In other words, if $S \in \mathbf{NP}$ and there exists a Karp reduction from S' to S, then $S' \in \mathbf{NP}$.

In other words, $\mathbf{NP}$ is downward-closed.

Proof:

Let $S \in \mathbf{NP}$, and so it has a polynomial proof system V, and let its polynomial be p. Let f be the Karp reduction from S' to S. We will define a verifier $V'(x,y) = V(f(x),y)$. Since f can be computed in polynomial time, and V runs in polynomial time, V' also runs in polynomial time.

And since f can be computed in polynomial time, there exists a polynomial q such that $|f(x)| \leq q(|x|)$ (since it only has polynomial time to add data). So if $x \in S'$ then $f(x) \in S$, and so there exists a y such that $|y| \leq p(|f(x)|)$ and $V(f(x),y) = 1$. So $V'(x,y) = 1$ where $|y| \leq p(|f(x)|) \leq p(q(|x|))$ which is a polynomial (we can assume that the polynomials are increasing, as we can assume that they are of the form x^n). And if $x \notin S'$ then $f(x) \notin S$ and so for every y, $V'(x,y) = V(f(x),y) = 0$ as required. ■

It is also obvious that $\mathbf{P}$ is closed under Karp reductions. If $S \in \mathbf{P}$ and f is a Karp reduction from S' to S, then if M solves S in polynomial time, then we simply return $M(f(x))$ and this solves S'.

Definition 6.3:

We define the class **coNP** to be

$$\mathbf{coNP} = \{S \mid S^c \in \mathbf{NP}\}$$

Note that since $\mathbf{P}$ is closed under complements, $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$. And if $\mathbf{P} = \mathbf{NP}$ then $\mathbf{NP}$ is closed under complements and so $\mathbf{NP} = \mathbf{coNP}$. So it is an open problem if $\mathbf{NP} \neq \mathbf{coNP}$.

Proposition 6.4:

If **coNP** contains an $\mathbf{NP}$-hard problem, then $\mathbf{NP} = \mathbf{coNP}$.

Proof:

Notice that **coNP** $\subseteq$ **NP** if and only if **NP** $\subseteq$ **coNP**. Since if **coNP** $\subseteq$ **NP** then if $S \in$ **NP**, $S^c \in$ **coNP** $\subseteq$ **NP** and so $S \in$ **coNP** as required. Similar for the converse.

Suppose S is **NP**-hard and in **coNP**. And since S^c is in **NP**, there exists a Karp reduction f from S^c to S, since S is **NP**-hard. Let S' be in **coNP**, then $(S')^c \in$ **NP** and so there exists a Karp reduction g from $(S')^c$ to S, and this is also a Karp reduction from S' to S^c. So for every $S' \in$ **coNP**, there exists a Karp reduction from S' to $S^c \in$ **NP**, and since **NP** is downward-closed, $S' \in$ **NP**. Thus **coNP** $\subseteq$ **NP**, and so **coNP** $=$ **NP**. ■

But **NP**-hard problems require that there exists a Karp reduction from every **NP** problem to them. Karp reductions are quite restrictive, what if we loosened this constraint to require Cook reductions instead?

Proposition 6.5:

If **NP** $\cap$ **coNP** contains a problem for which there exists a Cook reduction from every **NP** problem to it, then **NP** $=$ **coNP**.

Proof:

We will show **coNP** $\subseteq$ **NP**. Let S be such a problem in **NP** $\cap$ **coNP**. Let $S' \in$ **coNP**, and so there exists a Cook reduction from $(S')^c$ to S. We can take this Cook reduction and invert its output, and this defines a Cook reduction from S' to S.

So for every $S' \in$ **coNP**, there exists a polynomial-time algorithm A with an oracle for S which decides S'. Notice that A queries the oracle a polynomial number of times. And since $S \in$ **NP** $\cap$ **coNP** there exist verifiers V_S and V_{S^c} with polynomials p_S and p_{S^c}, which form polynomial proof systems for S and S^c respectively. Let us define a verifier V' which accepts as a witness a sequence of results of queries and witnesses for the oracle.

And so the input for V' is of the form $\big(x, \big((\sigma_1, w_1), (\sigma_2, w_2), \ldots, (\sigma_t, w_t)\big)\big)$ where σ_i is the result of a query for the oracle, and w_i is a witness. Then V' runs $A(x)$, and whenever A queries the oracle with a query of the form $q_i \in S$, V' will check,

(1) If $\sigma_i = 1$ then V' checks that $V_S(q_i, w_i) = 1$. If so, V' continues. If not, V' returns zero.

(2) If $\sigma_i = 0$ then V' checks that $V_{S^c}(q_i, w_i) = 0$. If so, V' continues. If not, V' returns zero.

V' runs a polynomial time algorithm, and at each step it may run another polynomial time algorithm (V_S or V_{S^c}), which all in all takes polynomial time.

If $x \in S'$ then let y be a sequence of answers to the oracle calls in A, along with the polynomial-length witnesses for V_S or V_{S^c}. Since A makes a polynomial number of oracle queries, y has a polynomial number of values, and each value is polynomial in x (the algorithm only has time to construct queries which are polynomial in x, and these have witnesses which are polynomial in their length, which is polynomial in the length of x).

Now, if $x \notin S'$ then for any sequence y if V' does not accept all the witnesses, then it returns zero. Otherwise, every oracle call gives a valid result (ie. $\sigma_i = 1$ means $q_i \in S$, and $\sigma_i = 0$ means $q_i \notin S$), and so V' acts exactly like A, and so V' would return zero.

Thus V' is a polynomial proof system for S', as required. ■