Computability and Complexity

Lecture 2, Thursday August 3, 2023

Ari Feiglin

For every search problem R, there exists a related decision problem

$$S_R = \{x \mid \exists y\colon (x,y) \in R\}$$

S_R essentially asks "does this input have a solution?"

Note:

Recall that in our definition of **PC** we similarly related a search problem with a decision problem. For a search problem R we instead had the decision problem

$$\{(x,y) \mid (x,y) \in R\}$$

(This is equal to R, but we are viewing this as a decision problem, not a search problem. Meaning that we don't view (x,y) as x being the input and y being the output, rather (x,y) is an element of the decision.)

This decision problem is more closely related to R, but it doesn't really simplify anything; after all the set is literally equal to R.

Proposition 2.1:

$\mathbf{P} = \mathbf{NP}$ if and only if $\mathbf{PC} \subseteq \mathbf{PF}$.

Proof:

Suppose $\mathbf{PC} \subseteq \mathbf{PF}$, then let $S \in \mathbf{NP}$. So there exists a verifier V which runs in polynomial time and a polynomial p such that $x \in S$ if and only if there exists a y where $|y| \leq p(|x|)$ and $V(x,y) = 1$. Then let us define the search problem R

$$R = \{(x,y) \mid |y| \leq p(|x|),\, V(x,y) = 1\}$$

then $R \in \mathbf{PC}$ since V solves R and runs in polynomial time. Thus $R \in \mathbf{PF}$, and so there exists an algorithm A such that for every x, $A(x) \in R(x)$, meaning $V(x, A(x)) = 1$ and $|A(x)| \leq p(|x|)$ (if such an $A(x)$ exists). And so let us define an algorithm B where $B(x) = 1$ if and only if $A(x) \neq \perp$.

Then B runs in polynomial time, and $B(x) = 1$ if and only if there exists an $A(x)$ such that $V(x, A(x)) = 1$ and $|A(x)| \leq p(|x|)$ which is if and only if $x \in S$. So B solves S in polynomial time, and so $S \in \mathbf{P}$.

Now suppose $\mathbf{P} = \mathbf{NP}$ and let $R \in \mathbf{PC}$. Then there exists an algorithm A which runs in polynomial time, and a polynomial p, such that $A(x,y) = 1$ if and only if $(x,y) \in R$. Let us define the *decision* problem

$$S'_R = \{(x,u) \mid \exists w\colon (x, uw) \in R\}$$

which asks if there exists a solution to x which starts with u. Then S'_R is in **NP** as $(x,u) \in S'_R$ if and only if there exists a w where $|u| + |w| \leq p(|x|)$ and $A(x, uw) = 1$. Thus $V((x,u), w) = A(x, uw)$ is a polynomial proof system for S'_R. And so $S'_R \in \mathbf{P}$, so there exists a polynomial time algorithm D where $D(x,u) = 1$ if and only if there exists a w where $(x, uw) \in R$.

Then let us define an algorithm B where we first run $D(x, \varepsilon)$, and if it returns zero then return $\perp$ (as there does not exist a solution). Otherwise, let us run $D(x, 1)$ and if it returns zero then the solution must start with zero, and so we run $D(x, 11)$ and so on.

function $C(x,u)$
    if $(A(x,u) = 1)$
        return u
    else if $(D(x, u1) = 1)$
        return $C(x, u1)$
    else
        return $C(x, u0)$

```
    end if
end function

function B(x)
    if (D(x, ε) = 0)  return ⊥
    else return C(x, ε)
end function
```

So our algorithm checks that $D(x,\varepsilon) \neq 0$, then it runs and returns $C(x,\varepsilon)$. There are at most $p(|x|)$ steps, so this runs in polynomial time. ■

Notice that in the proof above, we used a solution to one problem to solve another. This is called an *reduction*:

Definition 2.2:

If A and B are both problems, then a **reduction** from A to B means that A is no harder than B to solve (as in there exists a solution to A using B). This is denoted $A \leq B$.

So if $A \leq B$, and you do not know how to solve A, then you do not know how to solve B.

Definition 2.3:

A **Cook reduction** from a problem A to B is a polynomial time algorithm which solves A using an oracle for B. An **oracle** for B takes input and returns a solution for B within one step.

So for example, in the above proof we used a Cook reduction from R to S'_R.

Definition 2.4:

If R is a search problem, then a **self-reduction** is a Cook reduction from R to S_R.

Example 2.5:

Recall that

$$R_{\mathrm{SAT}} = \{(\varphi,\tau) \mid \varphi \text{ is a boolean formula in CNF, and } \varphi(\tau) \text{ is true}\}$$

We can create a self-reduction of R_{SAT}, which is a reduction from R_{SAT} to $S_{R_{\mathrm{SAT}}} = \mathrm{SAT}$.

Suppose we have an oracle A for SAT. And suppose φ is a formula over the set of variables $\{x_1,\dots,x_n\}$. What we do is first set x_1 to true, and then for each disjunction, it is of the form $\bigvee_{i=1}^n \varepsilon_i x_i$. If ε_1 is empty, then we can get rid of this disjunction, as it is true. Otherwise we remove x_1 from the disjunction. This forms a new formula in CNF φ_1, and we can check if φ_1 has a solution using our oracle for SAT. If it does, then we can recurse over φ_1 where the beginning of our solution has $x_1 = 1$.

Otherwise we form φ'_1 where we do a similar process but when $x_1 = 0$.

More explicitly,

(1) Ask the oracle if $\varphi \in \mathrm{SAT}$.

(2) If not, return $\perp$.

(3) Otherwise, set τ to ε.

(4) For $i = 1$ to n:

  (i) Define τ' to be $\tau 1$.

  (ii) Define φ' by valuating the first i variables as their elements in τ.

  (iii) Ask the oracle if $\varphi' \in \mathrm{SAT}$.

  (iv) If yes, set τ to be τ'.

  (v) Otherwise, set τ to be $\tau 0$.

Note:

If R is a search problem which is self-reducing (as in it has a self reduction), then R and S_R are equivalent up to a polynomial.

If we can solve R, then we can solve S_R by simply checking if the algorithm which solves R does not return $\perp$. And since there is a self-reduction, if we can solve S_R then we can solve R using a polynomial time algorithm (as Cook reductions are polynomial time).

Definition 2.6:

Let S_1 and S_2 be two decision problems. A **Karp reduction** from S_1 to S_2 is a function f which can be computed in polynomial time which satisfies

$$x \in S_1 \iff f(x) \in S_2$$

This is denoted $S_1 \leq_p^m S_2$ (p for polynomial, m for many-to-one, as f need not be injective).

Proposition 2.7:

If $S_2 \in \mathbf{P}$ and there is a Karp reduction from S_1 to S_2, then $S_1 \in \mathbf{P}$.

The proof is not too complicated: define an algorithm $A(x)$ which computes $f(x)$ and then determines if $f(x) \in S_2$. Since computing $f(x)$ and determining if $f(x) \in S_2$ both take polynomial time (since f is a Karp reduction, and $S_2 \in \mathbf{P}$), A is a polynomial time algorithm and therefore $S_2 \in \mathbf{P}$.