P, NP, and PH

1 Introduction to $\mathcal{NP}$

Recall the definition of the class $\mathcal{P}$:

Def 1.1 A is in P if there exists a Turing machine M and a polynomial p such that $\forall x$

- If $x \in A$ then $M(x) = YES$.
- If $x \notin A$ then $M(x) = NO$.
- For all x $M(x)$ runs in time $\leq p(|x|)$.

The typical way of defining NP is by using *non-deterministic* Turing machines. We will NOT be taking this approach. We will instead use quantifiers. This is equivalent to the definition using nondetermism.

Def 1.2 A is in NP if there exists a set $B \in$ P and a polynomial p such that

$$L = \{x \mid (\exists y)[|y| = p(|x|) \land (x, y) \in B]\}.$$

Here is some intution. Let $A \in$ NP.

- If $x \in A$ then there is a SHORT (poly in $|x|$) proof of this fact, namely y, such that x can be VERIFIED in poly time. So if I wanted to convince you that $x \in L$, I could give you y. You can verify $(x, y) \in B$ easily and be convinced.
- If $x \notin A$ then there is NO proof that $x \in A$.

2 NP **Completeness**

We first discuss the notion of *reductions*. We first define a Cook(-Turing) reduction:

Def 2.1 A *Cook(-Turing) reduction* from a language L to a language L' is a polynomial-time oracle machine M such that, if M' is any machine that decides L', then $M^{M'}$ decides L. We express the above by writing $L \leq_T^P L'$.

Another important, yet immediate, result is that (1) if there is a Cook reduction from L to L' and (2) $L' \in$ P, then $L \in$ P as well. Note, however, that this is not believed to be the case for languages in NP. For example, every coNP language is Cook-reducible to an NP language, but it is not believed that coNP $\subseteq$ NP.

A more restricted notion of a reduction is given next:

Def 2.2 A *Karp reduction* (also called a *many-to-one reduction*) from a language L to a language L' is a polynomial-time computable function f such that $x \in L$ iff $f(x) \in L'$. We express this by writing $L \leq_{\mathrm{m}}^{\mathrm{p}} L'$.

Note that any Karp reduction provides an immediate Cook reduction as well. However, here it is true that if there is a Karp reduction from L to L' and $L' \in \mathrm{NP}$, then $L \in \mathrm{NP}$.
It may be verified that all the above reductions are transitive.

2.1 Defining NP Completeness

With the above in place, we define NP-hardness and NP-completeness:

Def 2.3 A language L is NP-hard if for every language $L' \in \mathrm{NP}$, there is a Karp reduction from L' to L. A language L is NP-complete if it is NP-hard and also $L \in \mathrm{NP}$.

We remark that one could also define NP-hardness via *Cook* reductions. However, this seems to lead to a different definition. In particular, oracle access to any coNP-complete language is enough to decide NP, meaning that any coNP-complete language is NP-hard w.r.t. Cook reductions. On the other hand, if a coNP-complete language were NP-hard w.r.t. Karp reductions, this would imply NP = coNP (which is considered to be unlikely).

We show the "obvious" NP-complete language:

Theorem 2.4 *Define language L via:*

$$L = \left\{ \langle M, x, 1^t \rangle \mid \begin{array}{c} M \textit{ is a non-deterministic T.M.} \\ \textit{which accepts } x \textit{ within } t \textit{ steps} \end{array} \right\}.$$

Then L is NP*-complete.*

Proof: It is not hard to see that $L \in \mathrm{NP}$. Given $\langle M, x, 1^t \rangle$ as input, non-deterministically choose a legal sequence of up to t moves of M on input x, and accept if M accepts. This algorithm runs in non-deterministic polynomial time and decides L.

To see that L is NP-hard, let $L' \in \mathrm{NP}$ be arbitrary and assume that non-deterministic machine $M'_{L'}$ decides L' and runs in time n^c on inputs of size n. Define function f as follows: given x, output $\langle M'_{L'}, x, 1^{|x|^c} \rangle$. Note that (1) f can be computed in polynomial time and (2) $x \in L' \Leftrightarrow f(x) \in L$. We remark that this can be extended to give a Levin reduction (between R_L and $R_{L'}$, defined in the natural ways). ∎

3 More NP-Compete Languages

It will be nice to find more "natural" NP-complete languages. The *first* problem we prove NP-complete will have to use details of the machine model- Turing Machines. All later results will be reductions using known NP-complete problems.

Def 3.1 1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.

2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.

3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.

4. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals.

5. k-DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of exactly k literals.

Theorem 3.2 *CNFSAT is* NP*-complete.*

Proof: It is easy to see that $CNFSAT \in \mathrm{NP}$.
Let $L \in \mathrm{NP}$. We show that $L \leq_{\mathrm{m}}^{\mathrm{p}} CNFSAT$.
M be a TM and p, q be polynomials such that

$$L = \{x \mid (\exists y)[|y| = q(|x|) \text{ AND } M(x, y) = 1]\}$$

and $M(x, y)$ runs in time $q(|x| + |y|)$.
We will actually have to deal with the details of the M. Let $M = (Q, \Sigma, \delta, \Sigma, \delta, q_0, h)$
We will also need to represent what a Turing Machine is doing at every stage.
The machine itself has a tape, something like

$$\#abba\#ab@ab\#a$$

(We assume that everything to the right that is not seen is a #. Our convention is that you CANNOT go off to the left— from the left most symbol you can't go left.)
is in state q and the head is looking at (say) the @ sign.
We would represent this

$$\#abba\#ab(@, q)a$$

That is our convention— we extend the alphabet and allow symbols $\Sigma \times Q$. The symbol $(@, q)$ means the symbol is @, the state is q, and that square is where the head of the machine is.

If $x \in L$ then there is a y of length $q(|x|)$ such that the Turing machine on M accepts.

Lets us say that with more detail.

If $x \in L$ then there is a y and a sequence of configurations $C_1, C_2, \ldots, C_t$ such that

- C_1 is the configuration that says 'input is $x\#y$, and I am in the starting state.'
- For all i, C_{i+1} follows from C_i (note that M is deterministic) using δ.
- C_t is the configuration that says "END and output is 1"
- $t = p(|x| + q(|x|)$.

How to make all of this into a formula?

KEY 1: We will have a variable for every possible entry in every possible configuration. Hence the variables are $z_{i,j,\sigma}$ where $1 \le i, j \le t$, and $\sigma \in \Sigma \cup Q$. The intent is that if there is an accepting sequence of configurations then

$z_{i,j,\sigma} = T$ iff the j symbol in the ith configuration is σ.

To just make sure that for every i, j there is a unique σ such that $z_{i,j,\sigma} = T$ we have, for every $1 \le i \le j$, the following clauses.

$$\bigvee_{\sigma \in \Sigma \cup Q} z_{i,j,\sigma}$$

(NOTE- the actual formula would write out all of this and not be allowed to use $\bigvee$. With Poly time it MATTERS what kind of representation you use since we want computations to be poly time in the length of the input.)

for each $\sigma \in \Sigma \cup (\Sigma \times Q)$

$$z_{i,j,\sigma} \rightarrow \bigvee_{\tau \in (\Sigma \cup (\Sigma \times Q) - \{\sigma\}} \neg z_{i,j,\tau}$$

(It is an easy exercise to turn this into a set of clauses.)

KEY 2: The parts of the formula that say that C_1 is the starting configuration for $x\#y$ on the tape, and C_t is the configuration for saying DONE and output is 1, are both easy. Note that for the y part- WE DO NOT KNOW y. So we have to write that the y is a squence of elements of Σ of length $q(|x|)$.

Recall our convention for the first and last configuration:

Intuitively we start out with x and y laid out on the tape, and the head looking at the # just to the right of y. The machine then runs, and if it gets to the q_{accept} state then it accepts.

The following formula says that C_1 says 'start with x' Let $x = x_1 \cdots x_n$.

$$z_{1,1,x_1} \wedge \cdots z_{1,n,x_n} \wedge x_{1,n+1,\#} \wedge$$

$$\bigwedge_{i=n+2}^{n+q(|x|+1} \bigvee_{\sigma\in\Sigma} z_{1,i,\sigma}$$

$$\wedge z_{1,q(n)+n+2,(\#,s)} \wedge \bigwedge_{i=q(n)+n+3}^{t(n)} \wedge z_{1,i,\#}$$

Note that this formula is in CNF-form.

The following formula says that C_t says 'ends with accept'

$$\bigvee_{i=1}^{t(n)} \bigvee_{\sigma\in\Sigma} z_{t,i,(\sigma,q_{accept}}$$

KEY 3: How do we say that going from C_i you must goto C_{i+1}. We first do a thought experiment and then generalize. What if

$$\delta(q,a) = (p,b).$$

Then if the C_i says that you are in state q and looking at an a then C_{i+1} must be in state p and overwrite a with b. Note that in both cases *the rest of the configuration has not changed*.

How do we make this into a formula? The statement "C_i says that you are in state q and looking at an a" and the head is at the jth position is

$$z_{i,j,(a,q)}$$

We also have to know what else is around it. Assume that there is a b on the left and a c on the right. So we have

$$(z_{i,j-1,b} \wedge (z_{i,j,(a,q)} \wedge (z_{i,j+1,c}.$$

The statement that C_{i+1} is in state p and having overwritten a with b

$$(z_{i+1,j-1,b} \wedge (z_{i+1,j,(b,p)} \wedge (z_{i+1,j+1,c}.$$

This leads to the formula

$$\bigwedge_{i,j=1}^{t} (z_{i,j-1,b} \wedge (z_{i,j,(a,q)} \wedge (z_{i,j+1,c} \rightarrow (z_{i+1,j-1,b} \wedge (z_{i+1,j,(b,p)} \wedge (z_{i+1,j+1,c}.$$

This formula can be put into CNF-form.

For all of the δ values we need a similar formula.

PUTTING IT ALL TOGETHER

Take the $\wedge$ of the formulas in the last three KEY points and you have a formula ϕ

$$x \in L \iff \phi \in CNFSAT.$$

▮

4 Other NP-Complete Problems

Now that we have SAT is NP-Complete many other problems can be shown to be NP-complete. They come from many different areas of computer science and math: graph theory, scheduling, number theory, and others.

There are literally thousands of natural and distinct NP-complete problems!

5 Relating Function Problems to Decision Problems

Consider the NP-complete problem

$$CLIQUE = \{(G, k) \mid G \text{ has a clique of size } k\}.$$

Note that while this is a nice problem, its not quite the one we really want to solve. We want to compute the *function*

$SIZECLIQUE(G) = k$ such that k is the size of the largest clique in G.

Or we may want to compute

$FINDCLIQUE(G) =$ the largest clique in G (Note- this is ambiguous as there could be a tie. This can be resolved in several ways.)

How hard are these problems?

Theorem 5.1 *$CLIQUE$ and $FINDCLIQUE$ are Cook-equivalent. In particular*

1. *$CLIQUE$ can be solved with one query to $FINDCLIQUE$.*
2. *$FINDCLIQUE(G)$ can be computed with $\log n$ queries to $CLIQUE$*

Proof:

The first part is trivial.

We give an algorithm for the second part.

1. Input G
2. Ask $(G, n/2) \in CLIQUE$? If YES then ask $(G, 3n/4) \in CLIQUE$. If NO then ask $(G, n/4) \in CLIQUE$.
3. Continue using binary search until you get to the answer. This will take $\log n$ queries.

∎

The theorem above can be generalized to saying that if $L \in NP$ then the function associated to it (this can be done in several ways) is Cook Equivalent to L. Details will be on a HW.

6 The Polynomial Hierarchy

Recall (one of) the definitions of NP.

Def 6.1 $A \in$ NP if there exists a polynomial p and a polynomial predicate B such that
$A = \{x \mid (\exists y)[|y| \le p(|x|) \wedge B(x,y)]\}$.

What if we allowed more quantifiers? Then what happens?

Notation 6.2

1. The expression
   $A = \{x \mid (\exists^p y)[B(x,y)]\}$
   means that there is a polynomial p such that
   $A = \{x \mid (\exists y, |y| \le p(|x|))[B(x,y)]\}$.
2. The expression
   $A = \{x \mid (\forall^p y)[B(x,y)]$
   means that there is a polynomial p such that
   $A = \{x \mid (\forall y, |y| \le p(|x|))[B(x,y)]\}$.
3. The expression
   $A = \{x \mid (\forall^p y)(\exists^p z)[B(x,y,z)]$
   means that there are polynomials p_1, p_2 such that
   $A = \{x \mid (\forall y, |y| \le p_1(|x|))(\exists z, |z| \le p_2(|x|))[B(x,y,z)]\}$.
4. One can define this notation for as long a string of quantifiers as you like. We leave the formal definition to the reader.

In the following definition we include a definition and an alternative definition.

Def 6.3

1. $A \in \Sigma_0^p$ if $A \in$ P. $A \in \Pi_0^p$ if $A \in$ P. (We include this so we use it inductively later.)

2. $A \in \Sigma_1^p$ if there exists a set $B \in \mathrm{P}$ such that
   $A = \{x \mid (\exists^p y)[B(x,y)]\}$.
   This is just NP.

3. $A \in \Pi_1^p$ if there exists a set $B \in \mathrm{P}$ such that
   $A = \{x \mid (\forall^p y)[B(x,y)]\}$.
   This is just all sets A such that $\overline{A} \in \mathrm{NP}$. It is often called co-NP.

4. $A \in \Sigma_2^p$ if there exists a set $B \in \mathrm{P}$ such that
   $A = \{x \mid (\exists^p y)(\forall^p z)[B(x,y,z)]\}$.

5. $A \in \Sigma_2^p$ (alternative definition) if there exists a set $B \in \Pi_1^p$ such that
   $A = \{x \mid (\exists^p y)[B(x,y)]\}$.

6. $A \in \Pi_2^p$ if there exists a set $B \in \mathrm{P}$ such that
   $A = \{x \mid (\forall^p y)(\exists^p z)[B(x,y,z)]\}$.

7. $A \in \Pi_2^p$ (alternative definition) if $\overline{A} \in \Sigma_2^p$.

8. Let $i \in \mathsf{N}$. If i is even then $A \in \Sigma_i^p$ if there exists $B \in \mathrm{P}$ such that
   $A = \{x \mid (\exists^p y_1)(\forall^p y_2) \cdots (\forall^p y_i)[B(x,y_1,\ldots,y_i)]$
   If i is odd then $A \in \Sigma_i^p$ if there exists $B \in \mathrm{P}$ such that
   $A = \{x \mid (\exists^p y_1)(\forall^p y_2) \cdots (\exists^p y_i)[B(x,y_1,\ldots,y_i)]$

9. Let $i \in \mathsf{N}$. If i is even then $A \in \Pi_i^p$ if there exists $B \in \mathrm{P}$ such that
   $A = \{x \mid (\forall^p y_1)(\exists^p y_2) \cdots (\exists^p y_i)[B(x,y_1,\ldots,y_i)]$
   If i is odd then $A \in \Pi_i^p$ if there exists $B \in \mathrm{P}$ such that
   $A = \{x \mid (\forall^p y_1)(\exists^p y_2) \cdots (\forall^p y_i)[B(x,y_1,\ldots,y_i)]$

10. Let $i \in \mathsf{N}$ and $i \geq 1$. $A \in \Sigma_i^p$ (alternative definition) if there exists $B \in \Pi_{i-1}^p$ such that
    $A = \{x \mid (\exists^p y)[B(x,y)]\}$.
    (Note- we use the definition of Σ_0^p, Π_0^p here.)

11. $A \in \Pi_i^p$ (alternative definition) if $\overline{A} \in \Sigma_i^p$.

12. The *polynomial hierarchy*, denoted PH, is $\bigcup_{i=0}^{\infty} \Sigma_i^{\mathrm{p}}$. Note that this is the same as $\bigcup_{i=0}^{\infty} \Pi_i^{\mathrm{p}}$.

Def 6.4 A set A is Σ_i^p-*complete* if both of the following hold.

1. $A \in \Sigma_i^p$, and
2. For all $B \in \Sigma_i^p$, $B \le_{\mathrm{m}}^{\mathrm{p}} A$.

Def 6.5 A set A is Π_i^p-*complete* if both of the following hold.

1. $A \in \Pi_i^p$, and
2. For all $B \in \Pi_i^p$, $B \le_{\mathrm{m}}^{\mathrm{p}} A$.

Def 6.6 A set A is Π_i^p-*complete* (Alternative Definition) if $\overline{A}$ is Σ_i^p-complete.

Example 6.7 In all of the examples below x and y and x_i are vectors of Boolean variables.

1. $A = \{\phi(x,y) \mid (\exists b)(\forall c)[\phi(b,c)]\}$. This set is Σ_2^p-complete. It is clearly in Σ_2^p. This is called QBF_2. The QBF stands for Quantified Boolean Formula. The proof that it is Σ_2^{p}-complete uses Cook's Theorem.
2. One can define QBF_i easily. It is Σ_i^p-complete.
3. QBF is the set of all $\phi(x_1,\ldots,x_n)$ (the x_i's are vectors of variables) such that $(\exists x_1)(\forall x_2)\cdots(Qx_n)[\phi(x_1,\ldots,x_n)]$. ($Q$ is $\exists^p$ if n is odd and is $\forall^p$ if n is even.) This set is thought to not be in any Σ_i^p or Π_i^p.
4. Let $TWO = \{\phi \mid \phi$ has exactly two satisfying assignments $\}$. We show that $TWO \in \Sigma_2^p$.

   $TWO =$

   $\{\phi \mid (\exists b,c)(\forall d)[b \neq c \wedge \phi(b) \wedge \phi(c) \wedge (\phi(d) \rightarrow ((d=b) \vee (d=c)))\}$

   It is not known if TWO is Σ_2^p-complete; however it is thought to NOT be.
5. One can define $THREE$, $FOUR$, etc. easily. They are all in Σ_2^p.
6. One can define variants of TWO having to do with finding TWO Hamiltonian cycles, TWO k-cliques, etc. Also $THREE$, etc. These are all Σ_2^p.
7. $ODD = \{\phi \mid \phi$ has an odd number of satisfying assignments $\}$ is thought to NOT be in PH.

Recall that

There are literally thousands of natural and distinct NP-complete problems!

What about Σ_2^p-complete problems? Other levels? Alas- there are very few of these. So why do we care about PH ?

We think that $SAT \notin \mathrm{P}$ since

$$SAT \in \mathrm{P} \rightarrow \mathrm{P} = \mathrm{NP}.$$

We tend to think that PH does not collapse to a lower level of the hierarchy (e.g., that $\mathrm{PH} = \Sigma_2^{\mathrm{p}}$). Hence if we have a statement XXX that we do not think is true but cannot prove is false, we will be happy to instead show

$$XXX \rightarrow \text{PH collapses} .$$

7 Collapsing PH

Theorem 7.1 *If* $\Pi_1^{\mathrm{p}} \subseteq \Sigma_1^{\mathrm{p}}$ *then* $\mathrm{PH} = \Sigma_1^{\mathrm{p}} = \Pi_1^{\mathrm{p}}$.

Proof: Assume $\Sigma_1^{\mathrm{p}} = \Pi_1^{\mathrm{p}}$. We first show that $\Sigma_2^{\mathrm{p}} = \Sigma_1^{\mathrm{p}}$.

Let $L \in \Sigma_2^{\mathrm{p}}$. Hence there is a set $B \in \Pi_1^{\mathrm{p}}$ such that

$$L = \{x \mid (\exists^p y)[(x, y) \in B]\}.$$

Since $B \in \Pi_1^{\mathrm{p}}$, by the premise $B \in \Sigma_1^{\mathrm{p}}$. Therefore there exists $C \in \mathrm{P}$ such that

$$B = \{(x, y) \mid (\exists^p z)[(x, y, z) \in C]\}.$$

Replacing this definition of B in the definition of L we obtain

$$L = \{x \mid (\exists^p y)(\exists^p z)[(x, y, z) \in C]\}.$$

This is clearly in Σ_1^{p}. Hence $\Sigma_2^{\mathrm{p}} \subseteq \Sigma_1^{\mathrm{p}}$. Hence we have $\Sigma_2^{\mathrm{p}} = \Sigma_1^{\mathrm{p}}$. By complementing both sides we get $\Pi_2^{\mathrm{p}} = \Pi_1^{\mathrm{p}}$.

One can now easily show that, for all i, $\Sigma_i^{\mathrm{p}} = \Sigma_1^{\mathrm{p}}$ by induction. One then gets $\Pi_i^{\mathrm{p}} = \Pi_1^{\mathrm{p}}$. Hence $\mathrm{PH} = \Pi_1^{\mathrm{p}} = \Sigma_1^{\mathrm{p}}$. ∎

The following theorems are proven similarly

Theorem 7.2 *Let* $i \in \mathsf{N}$. *If* $\Pi_i^{\mathrm{p}} \subseteq \Sigma_i^{\mathrm{p}}$ *then* $\mathrm{PH} = \Sigma_i^{\mathrm{p}} = \Pi_i^{\mathrm{p}}$.

Theorem 7.3 *If* $\Sigma_i^{\mathrm{p}} \subseteq \Pi_i^{\mathrm{p}}$ *then* $\mathrm{PH} = \Sigma_i^{\mathrm{p}} = \Pi_i^{\mathrm{p}}$.