P, NP, and PH

1 Introduction to \( \mathcal{NP} \)

Recall the definition of the class \( \mathcal{P} \):

**Def 1.1** \( A \) is in \( \mathcal{P} \) if there exists a Turing machine \( M \) and a polynomial \( p \) such that

- 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 \( \mathcal{NP} \) 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 nondeterminism.

**Def 1.2** \( A \) is in \( \mathcal{NP} \) if there exists a set \( B \in \mathcal{P} \) and a polynomial \( p \) such that

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

Here is some intuition. Let \( A \in \mathcal{NP} \).

- 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

**Def 2.1** A **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 \( L \leq_{p} m L' \).

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 2.2** A language \( L \) is NP-hard if for every language \( L' \in \text{NP} \), there is a reduction from \( L' \) to \( L \). A language \( L \) is NP-complete if it is NP-hard and also \( L \in \text{NP} \).

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. reductions, this would imply \( \text{NP} = \text{coNP} \) (which is considered to be unlikely).

We show the “obvious” NP-complete language:

**Theorem 2.3** Define language \( L \) via:

\[
L = \left\{ \langle M, x, 1^t \rangle \mid M \text{ is a non-deterministic T.M. which accepts } x \text{ within } t \text{ steps} \right\}.
\]

Then \( L \) is NP-complete.

**Proof:** It is not hard to see that \( L \in \text{NP} \). 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 \( L' \in \text{NP} \) 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) \( x \in L' \iff f(x) \in L \). 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, \( \phi(\overline{x}) \in SAT \) if there exists a vector \( \overline{b} \) such that \( \phi(\overline{b}) = \text{TRUE} \).

2. CNF SAT is the set of all boolean formulas in SAT of the form \( C_1 \land \cdots \land C_m \) where each \( C_i \) is an \( \lor \) of literals.
3. *k*-SAT is the set of all boolean formulas in SAT of the form $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ of exactly $k$ literals.

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

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

The following was proven by Stephen Cook and Leonid Levin independently around 1970.

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

**Proof:** It is easy to see that CNFSAT $\in$ NP.

Let $L \in$ NP. We show that $L \leq_p^{c} CNFSAT$.

Let $M$ 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#
```

(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.

Let 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 $\delta$.  

3
• $C_t$ 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 $1 \leq i, j \leq t$, 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 $i$th configuration is $\sigma$.

To just make sure that for every $i,j$ there is a unique $\sigma$ such that $z_{i,j,\sigma} = T$ we have, for every $1 \leq i \leq j$, 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 sequence of elements of $\Sigma$ 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 $x = x_1 \cdots x_n$.

\[ z_{1,1,x_1} \land \cdots \land z_{1,n,x_n} \land x_{1,n+1,\#} \land \bigwedge_{i=n+2}^{n+q(|x|)+1} \bigvee_{\sigma \in \Sigma} z_{1,i,\sigma} \land z_{1,q(n)+n+2,\#} \land \bigwedge_{i=q(n)+n+3}^{t(n)} z_{1,i,\#} \]

Note that this formula is in CNF-form.

The following formula says that $C_t$ says ‘ends with 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 $j$th 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

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

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

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

This leads to the formula

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

This formula can be put into CNF-form.

For all of the $\delta$ values we need a similar formula.

PUTTING IT ALL TOGETHER

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

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

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

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

\[ \text{SIZECLIQUE}(G) = k \text{ such that } k \text{ is the size of the largest clique in } G. \]

Or we may want to compute

\[ \text{FINDCLIQUE}(G) = \text{the largest clique in } G \text{ (Note- this is ambiguous as there could be a tie. This can be resolved in several ways.)} \]

How hard are these problems?

*Theorem 5.1* \textit{CLIQUE} and \textit{FINDCLIQUE} are Cook-equivalent. In particular

1. \textit{CLIQUE} can be solved with one query to \textit{FINDCLIQUE}.

2. \textit{FINDCLIQUE}(G) can be computed with \( \log n \) queries to \textit{CLIQUE}

*Proof:*

The first part is trivial.

We give an algorithm for the second part.

1. Input \( G \)

2. Ask \((G, n/2) \in \text{CLIQUE} \)? If YES then ask \((G, 3n/4) \in \text{CLIQUE} \). If NO then ask \((G, n/4) \in \text{CLIQUE} \).

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** $A \in \text{NP}$ if there exists a polynomial $p$ and a polynomial predicate $B$ such that

\[ A = \{ x \mid (\exists y)[|y| \leq p(|x|) \land B(x, y)] \}. \]

What if we allowed more quantifiers? Then what happens?

**Notation 6.2**

1. The expression

\[ A = \{ x \mid (\exists y)[B(x, y)] \} \]

means that there is a polynomial $p$ such that

\[ A = \{ x \mid (\exists y, |y| \leq p(|x|))[B(x, y)] \}. \]

2. The expression

\[ A = \{ x \mid (\forall y)[B(x, y)] \]

means that there is a polynomial $p$ such that

\[ A = \{ x \mid (\forall y, |y| \leq p(|x|))[B(x, y)] \}. \]

3. The expression

\[ A = \{ x \mid (\forall y)(\exists z)[B(x, y, z)] \]

means that there are polynomials $p_1, p_2$ such that

\[ A = \{ x \mid (\forall y, |y| \leq p_1(|x|))(\exists z, |z| \leq p_2(|x|))[B(x, y, z)] \}. \]

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. $A \in \Sigma_p^0$ if $A \in \text{P}$. $A \in \Pi_p^0$ if $A \in \text{P}$. (We include this so we use it inductively later.)

2. $A \in \Sigma_p^1$ if there exists a set $B \in \text{P}$ such that

\[ A = \{ x \mid (\exists y)[B(x, y)] \}. \]

This is just NP.
3. $A \in \Pi^p_0$ if there exists a set $B \in P$ such that
   
   $A = \{x \mid (\forall y)\{B(x, y)\}\}$.

   This is just all sets $A$ such that $\overline{A} \in \text{NP}$. It is often called co-NP.

4. $A \in \Sigma^p_2$ if there exists a set $B \in P$ such that
   
   $A = \{x \mid (\exists y)(\forall z)\{B(x, y, z)\}\}$.

5. $A \in \Sigma^p_2$ (alternative definition) if there exists a set $B \in \Pi^p_1$ such that
   
   $A = \{x \mid (\exists y)\{B(x, y)\}\}$.

6. $A \in \Pi^p_2$ (alternative definition) if $A \in \Sigma^p_2$.

7. Let $i \in \mathbb{N}$. If $i$ is even then $A \in \Sigma^p_i$ if there exists $B \in P$ such that
   
   $A = \{x \mid (\exists y_1)(\forall y_2)\cdots(\forall y_i)\{B(x, y_1, \ldots, y_i)\}\}$

   If $i$ is odd then $A \in \Sigma^p_i$ if there exists $B \in P$ such that
   
   $A = \{x \mid (\exists y_1)(\forall y_2)\cdots(\exists y_i)\{B(x, y_1, \ldots, y_i)\}\}$

8. Let $i \in \mathbb{N}$. If $i$ is even then $A \in \Pi^p_i$ if there exists $B \in P$ such that
   
   $A = \{x \mid (\forall y_1)(\exists y_2)\cdots(\forall y_i)\{B(x, y_1, \ldots, y_i)\}\}$

   If $i$ is odd then $A \in \Pi^p_i$ if there exists $B \in P$ such that
   
   $A = \{x \mid (\forall y_1)(\exists y_2)\cdots(\exists y_i)\{B(x, y_1, \ldots, y_i)\}\}$

9. Let $i \in \mathbb{N}$. If $i$ is even then $A \in \Sigma^p_i$ (alternative definition) if there exists $B \in \Pi^p_{i-1}$ such that
   
   $A = \{x \mid (\exists y)\{B(x, y)\}\}$.

   (Note- we use the definition of $\Sigma^p_0$, $\Pi^p_0$ here.)

10. Let $i \in \mathbb{N}$ and $i \geq 1$. $A \in \Sigma^p_i$ (alternative definition) if there exists $B \in \Pi^p_{i-1}$ such that
    
    $A = \{x \mid (\exists y)\{B(x, y)\}\}$.

Def 6.4 A set $A$ is $\Sigma^p_i$-complete if both of the following hold.

1. $A \in \Sigma^p_i$, and

2. For all $B \in \Sigma^p_i$, $B \leq^p_m A$. 

12. The polynomial hierarchy, denoted PH, is $\bigcup_{i=0}^{\infty} \Sigma^p_i$. Note that this is the same as $\bigcup_{i=0}^{\infty} \Pi^p_i$. 

Def 6.5 A set $A$ is $\Pi^p_i$-complete if both of the following hold.
1. $A \in \Pi^p_i$, and
2. For all $B \in \Pi^p_i$, $B \leq^p_m A$.

Def 6.6 A set $A$ is $\Pi^p_i$-complete (Alternative Definition) if $\overline{A}$ is $\Sigma^p_i$-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 $\Sigma^p_2$-complete. It is clearly in $\Sigma^p_2$.
   This is called $QBF_2$. The $QBF$ stands for Quantified Boolean Formula. The proof that it is $\Sigma^p_2$-complete uses Cook-Levin Theorem.

2. One can define $QBF_i$ easily. It is $\Sigma^p_i$-complete.

3. $QBF$ is the set of all $\phi(x_1, \ldots, x_n)$ (the $x_i$’s are vectors of variables) such that
   $(\exists x_1)(\forall x_2) \cdots (Q x_n)[\phi(x_1, \ldots, x_n)]$. ($Q$ is $\exists^p$ if $n$ is odd and is $\forall^p$ if $n$ is even.)
   This set is thought to not be in any $\Sigma^p_i$ or $\Pi^p_i$.

4. Let $TWO = \{ \phi \mid \phi$ has exactly two satisfying assignments $\}$. We show that $TWO \in \Sigma^p_2$.
   $TWO = \{ \phi \mid (\exists b, c)(\forall d)[b \neq c \land \phi(b) \land \phi(c) \land (\phi(d) \rightarrow ((d = b) \lor (d = c)))\}$
   It is not known if $TWO$ is $\Sigma^p_2$-complete; however it is thought to NOT be.

5. One can define $THREE$, $FOUR$, etc. easily. They are all in $\Sigma^p_2$.

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 $\Sigma^p_2$.

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 $\Sigma^p_2$-complete problems? Other levels? Alas- there are very few of these. So why do we care about PH?
   We think that $SAT \notin P$ since
   
   $SAT \in P \rightarrow P = NP$.

   We tend to think that PH does not collapse to a lower level of the hierarchy (e.g.,
   that PH = $\Sigma^p_2$). 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

   $XXX \rightarrow PH$ collapses.
7 Collapsing PH

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

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

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

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

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

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

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

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

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

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

The following theorems are proven similarly

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

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