1.1 Polynomial Time Hierarchy

We first define the classes in the polynomial-time hierarchy.

Definition 1.1 For each integer \( i \), define the complexity class \( \Sigma^p_i \) to be the set of all languages \( L \) such that there is a polynomial time Turing machine \( M \) and a polynomial \( q \) such that

\[
x \in L \iff \exists y_1 \in \{0,1\}^{q(|x|)} \forall y_2 \in \{0,1\}^{q(|x|)} \ldots Q_i y_i \in \{0,1\}^{q(|x|)} . M(x, y_1, \ldots, y_i) = 1
\]

where

\[
Q_i = \begin{cases} 
\forall & \text{if } i \text{ is even} \\
\exists & \text{if } i \text{ is odd}
\end{cases}
\]

and define the complexity class \( \Pi^p_i \) to be the set of all languages \( L \) such that there is a polynomial time Turing machine \( M \) and a polynomial \( q \) such that

\[
x \in L \iff \forall y_1 \in \{0,1\}^{q(|x|)} \exists y_2 \in \{0,1\}^{q(|x|)} \ldots Q_i y_i \in \{0,1\}^{q(|x|)} . M(x, y_1, \ldots, y_i) = 1
\]

where

\[
Q_i = \begin{cases} 
\exists & \text{if } i \text{ is even} \\
\forall & \text{if } i \text{ is odd}
\end{cases}
\]

(It is probably more consistent with notations for other complexity classes to use the notation \( \Sigma_i^P \) and \( \Pi_i^P \) for the classes \( \Sigma^p_i \) and \( \Pi^p_i \) but the latter is more standard notation.)

The polynomial-time hierarchy is \( \text{PH} = \bigcup_k \Sigma_k^p = \bigcup_k \Pi_k^p \).

Observe that \( \Sigma^p_0 = \Pi^p_0 = P \), \( \Sigma^p_1 = \text{NP} \) and \( \Pi^p_1 = \text{coNP} \). Here are some natural problems in higher complexity classes.

**EXACT-CLIQUE** = \( \{(G, k) \mid \text{the largest clique in } G \text{ has size } k\} \in \Sigma^p_2 \cap \Pi^p_2 \)

since TM \( M \) can check one of its certificates is a \( k \)-clique in \( G \) and the other is not a \( k+1 \)-clique in \( G \).

**MINCIRCUIT** = \( \{C \mid C \text{ is a circuit that is not equivalent to any smaller circuit}\} \in \Pi^p_2 \)
since
\[ \langle C \rangle \in \text{MINCircuit} \iff \forall \langle C' \rangle \exists y \text{ s.t. } (\text{size}(C') \geq \text{size}(C) \lor C'(y) \neq C(y)). \]

It is still open if \text{MINCircuit} is in \Sigma_2^P or if it is \Pi_2^P-complete. However, Umans [1] has shown that the analogous problem \text{MINDNF} is \Pi_2^P-complete (under polynomial-time reductions).

Define
\[ \Sigma_i\text{SAT} = \{ \langle \varphi \rangle \mid \varphi \text{ is a Boolean formula s.t. } \exists y_1 \in \{0,1\}^n \forall y_2 \in \{0,1\}^n \ldots Q_i y_i \in \{0,1\}^n \varphi(y_1, \ldots, y_i) \text{ is true} \}. \]

and define \Pi_i\text{SAT} similarly. Theorem we can convert the Turing machine computation into a Boolean formula and show that \Sigma_i\text{SAT} is \Sigma_i\text{P}-complete and \Pi_i\text{SAT} is \Pi_i\text{P}-complete.

It is generally conjectured that \forall i, \Sigma_i \neq \Pi_i.

**Lemma 1.2** \( \Pi_i \subseteq \Sigma_i \) implies that \( \Sigma_i \Sigma_i \Pi_i = \Pi_i \).

### 1.1.1 Alternative definition in terms of oracle TMs

**Definition 1.3** An oracle TM \( M^2 \) is a Turing machine with a separate oracle input tape, oracle query state \( q_{\text{query}} \), and two oracle answer states, \( q_{\text{yes}} \) and \( q_{\text{no}} \). The content of the oracle tape at the time that \( q_{\text{query}} \) is entered is given as a query to the oracle. The cost for an oracle query is a single time step. If answers to oracle queries are given by membership in a language \( A \), then we refer to the instantiated machine as \( M^A \).

**Definition 1.4** Let \( \mathcal{P}^A = \{ L(M^A) \mid M^2 \text{ is a polynomial-time oracle TM} \} \), let \( \mathcal{NP}^A = \{ L(M^A) \mid M^2 \text{ is a polynomial-time oracle NTM} \} \), and \( \text{coNP}^A = \{ \overline{L} \mid L \in \mathcal{NP}^A \} \).

**Theorem 1.5** For \( i \geq 0 \), \( \Sigma_i^p \subseteq \mathcal{NP}^{\Pi_i^p} \) (= \( \mathcal{NP}^{\Sigma_i^p} \)).

**Proof** \( \Sigma_i^p \subseteq \mathcal{NP}^{\Pi_i^p} \): The oracle NTM simply guesses \( y_1 \) and asks \( (x, y_1) \) for the \( \Pi_i^p \) oracle for \( \forall y_2 \in \{0,1\}^{|x|}, \ldots, Q_i y_i \in \{0,1\}^{|x(i)|}, M(x, y_1, y_2, \ldots, y_i) = 1 \).

\( \mathcal{NP}^{\Pi_i^p} \subseteq \Sigma_i^p \): Given a polynomial-time oracle NTM \( M^2 \) and a \( \Pi_i^p \) language \( A \) then \( x \in L = L(M^A) \) if and only if there is an accepting path of \( M^A \) on input \( x \).

To describe this accepting path we need to include a string \( y \) consisting of

- the polynomial length sequence of nondeterministic moves of \( M^2 \),
- the answers \( b_1, \ldots, b_m \) to each of the oracle queries during the computation,
- the queries \( z_1, \ldots, z_m \) given to \( A \) during the computation,

(Note that each of \( z_1, \ldots, z_m \) is actually determined by a deterministic polynomial time computation given the nondeterministic guesses and prior oracle answers so this can be checked at the end.) However, we need to ensure that each oracle answer \( b_i \) is actually the answer that the oracle \( A \) would return on inputs \( z_i \).

If all the answers \( b_i \) were \text{yes} answers then after an existential quantifier for \( y_1 = y \) we could simply check that \( (z_1, \ldots, z_m) \) are the correct queries by checking that they are in \( A^p \) which is in \( \Pi_i^p \) since \( A \in \Pi_i^p \).
The difficulty is that verifying the no answers is a $\Sigma_p^i$ problem (which likely can’t be expressed in $\Pi_p^i$). The trick to handle this is that since $A \in \Sigma_p^i$, there is some $B \in \Pi_p^{i-1} \subseteq \Pi_p^i$ and polynomial $p$ such that $z_j \notin A$ iff $\exists y'_j \in \{0,1\}^{p(|x|)} . (z_j, y'_j) \in B$.

Therefore, to express $L$ using a existentially quantified variable $y_1$ that includes $y$ as well as all $y'_j$ such that the query answer $b_j$ is no. It follows that $x \in L$ iff $\exists y_1, (x, y_1) \in A'$ for some $\Pi_p^i$ set $A'$ and thus $L \in \Sigma_p^{i+1}$.

It follows also that $\Pi_p^{i+1} = \text{coNP}^{\Sigma_p^i}$ for $i \geq 0$. This naturally also suggests the definition:

$$\Delta_p^0 = P$$
$$\Delta_p^{i+1} = P^{\Sigma_p^i} \quad \text{for } i \geq 0.$$  

Observe that $\Delta_p^i \subseteq \Sigma_p^i \cap \Pi_p^i$ and

$$\Delta_p^1 = P^P = P$$
$$\Sigma_p^1 = \text{NP}^P = \text{NP}$$
$$\Pi_p^1 = \text{coNP}^P = \text{coNP}$$
$$\Delta_p^2 = P^{NP} = P^{\text{SAT}} \supseteq \text{coNP}$$
$$\Sigma_p^2 = \text{NP}^{NP}$$
$$\Pi_p^2 = \text{coNP}^{NP}.$$  

Also, observe that in fact $\text{EXACTCLIQUE}$ is in $\Delta_2^p = P^{NP}$ by querying $\text{CLIQUE}$ on $\langle G, k \rangle$ and $\langle G, k + 1 \rangle$. 

Figure 1.1: The First Levels of the Polynomial-Time Hierarchy
1.2 Non-uniform Complexity

1.2.1 Circuit Complexity

Let $\mathbb{B}_n = \{f \mid f : \{0,1\}^n \rightarrow \{0,1\}\}$. A basis $\Omega$ is a subset of $\bigcup_n \mathbb{B}_n$.

**Definition 1.6** A Boolean circuit over basis $\Omega$ is a finite directed acyclic graph $C$ each of whose nodes is either

1. a source node labelled by either an input variable in $\{x_1, x_2, \ldots\}$ or constant $\in \{0,1\}$, or
2. a node of in-degree $d > 0$ called a *gate*, labelled by a function $g \in B_d \cap \Omega$.

There is a sequence of designated output gates (nodes). Typically there will just be one output node. Circuits can also be defined as straight-line programs with a variable for each gate, by taking a topological sort of the graph and having each line describes how the value of each variable depends on its predecessors using the associated function.

Say that Circuit $C$ is defined on $\{x_1, x_2, \ldots, x_n\}$ if its input variables $\subseteq \{x_1, x_2, \ldots, x_n\}$. $C$ defined on $\{x_1, x_2, \ldots, x_n\}$ computes a function in the obvious way, producing an output bit vector (or just a single bit) in the order of the output gate sequence.

Typically the elements of $\Omega$ we use are symmetric. Unless otherwise specified $\Omega = \{\land, \lor, \neg\} \subseteq B_1 \cup B_2$.

**Definition 1.7** A circuit family $C$ is an infinite sequence of circuits $\{C_n\}_{n=0}^\infty$ such that $C_n$ is defined on $\{x_1, x_2, \ldots, x_n\}$

- $\text{size}(C_n) =$ number of nodes in $C_n$.
- $\text{depth}(C_n) =$ length of the longest path from input to output.

A circuit family $C$ has size $S(n)$, depth $d(n)$, iff for each $n$

$$\text{size}(C_n) \leq S(n)$$
$$\text{depth}(C_n) \leq d(n)$$

We say that $A \in \text{SIZE}_\Omega(S(n))$ if there exists a circuit family of size $S(n)$ that computes $A$. Similarly we define $A \in \text{DEPTH}_\Omega(d(n))$. When we have the De Morgan basis we drop the subscript $\Omega$. Note that if another (complete) basis $\Omega$ is finite then it can only impact the size of circuits by a constant factor since any gate with fan-in $d$ can be simulated by a CNF formula of size $d2^d$. We write $\text{POLYSIZE} = \bigcup_k \text{SIZE}(n^k + k)$.

There are undecidable problems in $\text{POLYSIZE}$. In particular

$$\{1^n \mid \text{Turing machine } M_n \text{ accepts } \langle M_n \rangle \} \in \text{SIZE}(1)$$

as is any unary language.

Next time we will prove the following theorem due to Karp and Lipton:

**Theorem 1.8 (Karp-Lipton)** If $\text{NP} \subseteq \text{POLYSIZE}$ then $\text{PH} = \Sigma^p_2 \cap \Pi^p_2$.

**References**