Lesson 5: The polynomial hierarchy and the complexity classes for counting

Theme: The polynomial time hierarchy and the complexity classes for counting problems.

1 The polynomial hierarchy

For every integer \( i \geq 1 \), the class \( \Sigma^p_i \) is defined as follows. A language \( L \subseteq \{0,1\}^* \) is in \( \Sigma^p_i \), if there is a polynomial \( q(n) \) and a polynomial time DTM \( \mathcal{M} \) such that for every \( w \in \{0,1\}^* \), \( w \in L \) if and only if the following holds.

\[
\exists y_1 \in \{0,1\}^{q(|w|)} \forall y_2 \in \{0,1\}^{q(|w|)} \cdots Qy_i \in \{0,1\}^{q(|w|)} \mathcal{M} \text{ accepts } (w,y_1,\ldots,y_i) \tag{1}
\]

Here \( Q = \exists \), if \( i \) is odd and \( Q = \forall \), if \( i \) is even.

The class \( \Pi^p_i \) is defined as above, but the sequence of quantifiers in (1) starts with \( \forall \). Alternatively, it can also be defined as \( \Pi^p_i = \{ \overline{L} : L \in \Sigma^p_i \} \). Note that \( \text{NP} = \Sigma^p_1 \) and \( \text{coNP} = \Pi^p_1 \).

Remark 5.1 The class \( \Sigma^p_i \) can also be defined as follows. A language \( L \) is in \( \Sigma^p_i \), if there is a polynomial time ATM \( \mathcal{M} \) that decides \( L \) such that for every input word \( w \in \{0,1\}^* \), the run of \( \mathcal{M} \) on \( w \) can be divided into \( i \) layers. Each layer consists of nodes of the same depth in the run. (Recall that the run of an ATM is a tree.) In the first layer all nodes are labeled with existential configurations, in the second layer with universal configurations, and so on. It is not difficult to show that this definition is equivalent to the one above.

The polynomial time hierarchy (or, in short, polynomial hierarchy) is defined as the following class.

\[
\text{PH} \quad \text{def} \quad \bigcup_{i=1}^{\infty} \Sigma^p_i
\]

Note that \( \text{PH} \subseteq \text{PSPACE} \).

It is conjectured that \( \Sigma^p_1 \subseteq \Sigma^p_2 \subseteq \Sigma^p_3 \subseteq \cdots \). In this case, we say that the polynomial hierarchy does not collapse. We say that the polynomial hierarchy collapses, if there is \( i \) such that \( \text{PH} = \Sigma^p_i \), in which case we also say that the polynomial hierarchy collapses to level \( i \).

We define the notion of hardness and completeness for each \( \Sigma^p_i \) as follows. For \( i \geq 1 \), a language \( K \) is \( \Sigma^p_i \)-hard, if for every \( L \in \Sigma^p_i \), \( L \leq_p K \). It is \( \Sigma^p_i \)-complete, if it is in \( \Sigma^p_i \) and it is \( \Sigma^p_i \)-hard. The same notion can be defined analoguously for \( \text{PH} \) and each \( \Pi^p_i \).

Define the language \( \Sigma_i\text{-SAT} \) as consisting of true QBF of the form:

\[
\exists x_1 \forall x_2 \cdots Qx_i \varphi(x_1,\ldots,x_i)
\]

where \( \varphi(x_1,\ldots,x_i) \) is quantifier-free Boolean formula and \( Q = \exists \), if \( i \) is odd, and \( Q = \forall \), if \( i \) is even. Here \( x_1,\ldots,x_i \) are all vectors of boolean variables. In other words, \( \Sigma_i\text{-SAT} \) is a subset of \( \text{TQBF} \) where the number of quantifier alternation is limited to \( (i-1) \). The language \( \Pi_i\text{-SAT} \) is defined analoguously with the starting quantifiers being \( \forall \).

Theorem 5.2

- For every \( i \geq 1 \), \( \Sigma_i\text{-SAT} \) is \( \Sigma^p_i \)-complete and \( \Pi_i\text{-SAT} \) is \( \Pi^p_i \)-complete.
- If \( \Sigma^p_i = \Pi^p_i \) for some \( i \geq 1 \), then the polynomial hierarchy collapses.
- If there is language that is \( \text{PH} \)-complete, then the polynomial hierarchy collapses.
2 Complexity classes for counting problems

2.1 The class FP

We denote by $FP$ the class of functions $f : \{0, 1\}^* \rightarrow \mathbb{N}$ computable by polynomial time DTM. Here the convention is that a natural number is always represented in binary form. So, when we say that a DTM $M$ computes a function $f : \{0, 1\}^* \rightarrow \mathbb{N}$, on input word $w$, the output of $M$ on $w$ is $f(w)$ in the binary representation.

Let $\#CYCLE$ be the following problem.

<table>
<thead>
<tr>
<th>$#CYCLE$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A directed graph $G$.</td>
</tr>
<tr>
<td><strong>Task:</strong> Output the number of cycles in $G$.</td>
</tr>
</tbody>
</table>

As before, $\#CYCLE$ can also be viewed as a function. Note also that the number of cycles in a graph with $n$ vertices is at most exponential in $n$, thus, its binary representation only requires polynomially many bits.

**Theorem 5.3** If $\#CYCLE$ is in $FP$, then $P = NP$.

**Proof.** Let $G$ be a (directed) graph with $n$ vertices. We construct a graph $G'$ obtained by replacing every edge $(u, v)$ in $G$ with the following gadget:

![Graph Diagram]

Note that every simple cycle in $G$ of length $\ell$ becomes $(2m)^{\ell}$ cycles in $G'$. Now, let $m \overset{\text{def}}{=} n \log n$.

It is not difficult to show that $G$ has a hamiltonian cycle (i.e., a simple cycle of length $n$) if and only if $G'$ has more than $n^{(n^2)}$ cycles. So, if $\#CYCLE \in FP$, then checking hamiltonian cycle can be done is in $P$. ■

Note that checking whether a graph has a cycle itself can be done in polynomial time. However, as Theorem 5.3 above states, it is unlikely that counting the number of cycles can be done in polynomial time.

2.2 The class $\#P$

**Definition 5.4** A function $f : \{0, 1\}^* \rightarrow \mathbb{N}$ is in $\#P$, if there is a polynomial $q(n)$ and a polynomial time DTM $M$ such that for every word $w \in \{0, 1\}^*$, the following holds.

$$f(w) = |\{y : M \text{ accepts } (w, y) \text{ and } y \in \{0, 1\}^{q(|w|)}\}|$$

Alternatively, we can say that $f$ is in $\#P$, if there is a polynomial time NTM $M$ such that for every word $w \in \{0, 1\}^*$, $f(w)$ = the number of accepting runs of $M$ on $w$.

For a function $f : \{0, 1\}^* \rightarrow \mathbb{N}$, the language associated with the function $f$, denoted by $O_f$, is defined as $O_f \overset{\text{def}}{=} \{(w, i) : \text{the } i^{\text{th}} \text{ bit of } f(w) \text{ is } 1\}$. When we say that a TM $M$ has oracle access to a function $f$, we mean that it has oracle access to the language $O_f$.

We define $FP^f$ as the class of functions $g : \{0, 1\}^* \rightarrow \mathbb{N}$ computable by a polynomial time DTM with oracle access to $f$. 

2/3
Definition 5.5 Let \( f : \{0, 1\}^* \to \mathbb{N} \) be a function.

- \( f \) is \( \#P\)-hard, if \( \#P \subseteq \text{FP}^f \), i.e., every function in \( \#P \) is computable by a polynomial time DTM with oracle access to \( f \).

- \( f \) is \( \#P\)-complete, if \( f \in \#P \) and \( f \) is \( \#P\)-hard.

Let \( \#\text{SAT} \) be the following problem.

\[ \begin{array}{l}
\text{Input:} \quad \text{A boolean formula } \varphi. \\
\text{Task:} \quad \text{Output the number of satisfying assignments for } \varphi.
\end{array} \]

As before, the output numbers are to be written in binary form. We can also view \( \#\text{SAT} \) as a function \( \#\text{SAT} : \{0, 1\}^* \to \mathbb{N} \), where \( \#\text{SAT}(\varphi) = \) the number of satisfying assignment for \( \varphi \).

Theorem 5.6 \( \#\text{SAT} \) is \( \#P\)-complete.

Proof. Cook-Levin reduction (to prove the NP-hardness of SAT) is parsimonious. ■

There are usually two ways to prove a certain function is \( \#P\)-hard, as stated in Remark 5.7 and 5.8 below.

Remark 5.7 Let \( f_1 \) and \( f_2 \) be functions from \( \{0, 1\}^* \) to \( \mathbb{N} \). Suppose \( L_1 \) and \( L_2 \) be languages in \( \text{NP} \) such that \( f_1 \) and \( f_2 \) are the functions for the number of certificates for \( L_1 \) and \( L_2 \), respectively. That is, for every word \( w \in \{0, 1\}^* \),

\[ f_i(w) = \text{the number of certificates of } w \text{ in } L_i, \quad \text{for } i = 1, 2. \]

If \( f_1 \) is \( \#P\)-hard and there is a parsimonious (polynomial time) reduction from \( L_1 \) to \( L_2 \), then \( f_2 \) is \( \#P\)-hard.

Remark 5.8 Let \( f \) and \( g \) be two functions from \( \{0, 1\}^* \) to \( \mathbb{N} \). If \( f \) is \( \#P\)-hard and \( f \in \text{FP}^g \), then \( g \) is \( \#P\)-hard.

Since there is a parsimonious reduction from SAT to 3-SAT, by Theorem 5.6 and Remark 5.7, we have the following corollary.

Corollary 5.9 \( \#3\text{-SAT} \) is \( \#P\)-complete.

Corollary 5.9 can also be proved by showing \( \#\text{SAT} \in \text{FP}^{\#3\text{-SAT}} \).