Lecture 1

Polynomial Time Hierarchy

April 1, 2008
Lecturer: Paul Beame
Notes:

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 Σ_i^p to be the set of all languages L such that there is a polynomial time Turing machine M and a polynomial q such that

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

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 Π_i^p to be the set of all languages L such that there is a polynomial time Turing machine M and a polynomial q such that

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

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 $\Sigma_i\mathsf{P}$ and $\Pi_i\mathsf{P}$ for the classes Σ_i^p and Π_i^p but the latter is more standard notation.)

The *polynomial-time hierarchy* is $\mathsf{PH} = \bigcup_k \Sigma_k^p = \bigcup_k \Pi_k^p$.

Observe that $\Sigma_0^p = \Pi_0^p = \mathsf{P}$, $\Sigma_1^p = \mathsf{NP}$ and $\Pi_1^p = \mathsf{coNP}$. Here are some natural problems in higher complexity classes.

$$\text{EXACT-CLIQUE} = \{\langle G, k\rangle \mid \text{the largest clique in } G \text{ has size } k\} \in \Sigma_2^p \cap \Pi_2^p$$

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.

$$\text{MINCIRCUIT} = \{\langle C\rangle \mid \text{ C is a circuit that is not equivalent to any smaller circuit}\} \in \Pi_2^p$$

since

$$\langle C\rangle \in \textsc{MinCircuit} \Leftrightarrow \forall\langle C'\rangle\exists y\ s.t.\ (size(C') \geq size(C) \vee C'(y) \neq C(y)).$$

It is still open if $\textsc{MinCircuit}$ is in Σ_2^p or if it is Π_2^p-complete However, Umans [1] has shown that the analogous problem $\textsc{MinDNF}$ is Π_2^p-complete (under polynomial-time reductions).

Define

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

and define Π_iSAT similarly. Theorem we can convert the Turing machine computation into a Booleam formula and show that Σ_iSAT is Σ_i-complete and Π_iSAT is Π_i-complete.

It is generally conjectured that $\forall i, \mathsf{PH} \neq \Sigma_i^p$.

Lemma 1.2 $\Pi_i^p \subseteq \Sigma_i^p$ implies that $\mathsf{PH} = \Sigma_i^p = \Pi_i^p$.

1.1.1 Alternative definition in terms of oracle TMs

Definition 1.3 An oracle TM $M^?$ is a Turing machine with a separate oracle input tape, oracle query state q_{query}, and two oracle answer states, q_{yes} and q_{no}. The content of the oracle tape at the time that q_{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 $\mathsf{P}^A = \{L(M^A) \mid M^? \text{ is a polynomial-time oracle TM}\}$, let $\mathsf{NP}^A = \{L(M^A) \mid M^? \text{ is a polynomial-time oracle NTM}\}$, and $\mathsf{coNP}^A = \{\overline{L} \mid L \in \mathsf{NP}^A\}$.

Theorem 1.5 For $i \geq 0$, $\Sigma_{i+1}^p = \mathsf{NP}^{\Pi_i^p}$ $(= \mathsf{NP}^{\Sigma_i^p})$.

Proof $\Sigma_{i+1}^p \subseteq \mathsf{NP}^{\Pi_i^p}$: The oracle NTM simply guesses y_1 and asks (x, y_1) for the Π_i^p oracle for $\forall y_2 \in \{0,1\}^{q(|x|)} \ldots Q_{i+1}y_{i+1} \in \{0,1\}^{q(|x|)}.M(x, y_1, y_2, \ldots, y_{i+1}) = 1$.

$\mathsf{NP}^{\Pi_i^p} \subseteq \Sigma_{i+1}^p$: Given a polynomial-time oracle NTM $M^?$ and a Π_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 $M^?$,
- 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 *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^m which is in Π_i^p since $A \in \Pi_i^p$.

The difficulty is that verifying the *no* answers is a Σ_i^p problem (which likely can't be expressed in Π_i^p). The trick to handle this is that since $\overline{A} \in \Sigma_i^p$, there is some $B \in \Pi_{i-1}^p \subseteq \Pi_i^p$ 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 existentiallly 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 Π_i^p set A' and thus $L \in \Sigma_{i+1}^p$. $\square$

It follows also that $\Pi_{i+1}^p = \mathsf{coNP}^{\Sigma_i^p}$ for $i \geq 0$. This naturally also suggests the definition:

$$\begin{aligned} \Delta_0^p &= \mathsf{P} \\ \Delta_{i+1}^p &= \mathsf{P}^{\Sigma_i^p} \qquad \text{for } i \geq 0. \end{aligned}$$

Observe that $\Delta_i^p \subseteq \Sigma_i^p \cap \Pi_i^p$ and

$$\begin{aligned} \Delta_1^p &= \mathsf{P}^{\mathsf{P}} = \mathsf{P} \\ \Sigma_1^p &= \mathsf{NP}^{\mathsf{P}} = \mathsf{NP} \\ \Pi_1^p &= \mathsf{coNP}^{\mathsf{P}} = \mathsf{coNP} \\ \Delta_2^p &= \mathsf{P}^{\mathsf{NP}} = \mathsf{P}^{SAT} \supseteq \mathsf{coNP} \\ \Sigma_2^p &= \mathsf{NP}^{\mathsf{NP}} \\ \Pi_2^p &= \mathsf{coNP}^{\mathsf{NP}}. \end{aligned}$$

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

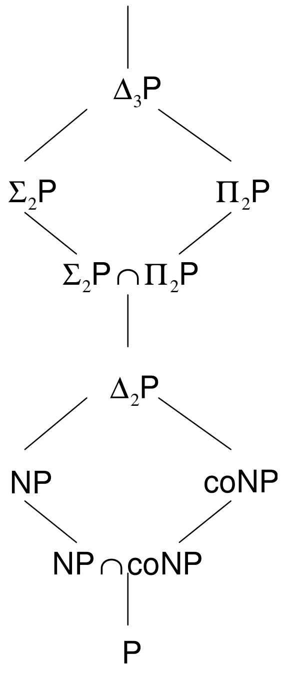

Figure 1.1: The First Levels of the Polynomial-Time Hierarchy

1.2 Non-uniform Complexity

1.2.1 Circuit Complexity

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

Definition 1.6 A Boolean circuit over basis Ω 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 $g \in \mathbb{B}_d \cap \Omega$.

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 Ω we use are symmetric. Unless otherwise specified $\Omega = \{\wedge, \vee, \neg\} \subseteq \mathbb{B}_1 \cup \mathbb{B}_2$.

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 $\{x_1, x_2, \ldots, x_n\}$

$size(C_n)$ = number of nodes in C_n.

$depth(C_n)$ =length of the longest path from input to output.

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

$$\begin{aligned} size(C_n) &\leq S(n) \\ depth(C_n) &\leq d(n) \end{aligned}$$

We say that $A \in \mathsf{SIZE}_\Omega(S(n))$ if there exists a circuit family of size $S(n)$ that computes A. Similarly we define $A \in \mathsf{DEPTH}_\Omega(d(n))$. When we have the De Morgan basis we drop the subscxript Ω. Note that if another (complete) basis Ω 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 $\mathsf{POLYSIZE} = \bigcup_k \mathsf{SIZE}(n^k + k)$.

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

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

as is any unary language.

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

Theorem 1.8 (Karp-Lipton) If $\mathsf{NP} \subseteq \mathsf{POLYSIZE}$ then $\mathsf{PH} = \Sigma_2^p \cap \Pi_2^p$.

References

[1] C. Umans. The minimum equivalant dnf problem and shortest implicants. *Journal of Computer and System Sciences*, 63(4):597–611, 2001.

Lecture 2

Relation of Polynomial-time Hierarchy, Circuits, and Randomized Computation

April 3, 2008
Lecturer: Paul Beame
Notes:

2.1 Turing Machines with Advice

Last lecture introduced non-uniformity through circuits. An alternate view of non-uniformity is Turing machines with an advice tape. The advice tape contains some extra information that depends only on the length of the input; i.e., on input x the TM gets $(x, \alpha_{|x|})$.

Definition 2.1 $\mathsf{TIME}(t(n))/f(n) = \{A \mid A$ is decided in time $O(t(n))$ by a TM with advice sequence $\{\alpha_n\}_n$ such that $\alpha_n \in \{0,1\}^{f(n)}\}$. (Note that the we ignore constant factors in the running time but not the advice.)

Now we can define the class of languages decidable in polynomial time with polynomial advice:

Definition 2.2 $\mathsf{P/poly} = \bigcup_{k,\ell} \mathsf{TIME}(n^k)/n^\ell$

Lemma 2.3 $\mathsf{P/poly} = \mathsf{POLYSIZE}$.

Proof $\mathsf{POLYSIZE} \subseteq \mathsf{P/poly}$: Given a polynomial-size circuit family $\{C_n\}_n$ produce a $\mathsf{P/poly}$ TM M by using advice strings $\alpha_n = \langle C_n \rangle$. On input x, M can then evaluate circuit $C_{|x|}$ on input x in time polynomial in $|x|$ and $|\langle C_{|x|}\rangle|$ which is polynomial in $|x|$.

$\mathsf{P/poly} \subseteq \mathsf{POLYSIZE}$: Given a $\mathsf{P/poly}$ TM M with advice strings $\{\alpha_n\}_n$, use the tableau construction from the Cook-Levin Theorem to construct a polynomial size circuit family with the advice strings hard-coded in the circuit. $\square$

If $\mathsf{NP} \subseteq \mathsf{P}$ then $\mathsf{PH} = \mathsf{P}$ but although $\mathsf{P/poly}$ contains undecidable languages we still get a collapse of the polynomial-time hierarchy if $\mathsf{NP} \subseteq \mathsf{P/poly}$.

Theorem 2.4 (Karp-Lipton) If $\mathsf{NP} \subseteq \mathsf{P/poly}$ then $\mathsf{PH} = \Sigma_2^p \cap \Pi_2^p$.

Proof Assume that $\mathsf{NP} \subseteq \mathsf{P}/\mathsf{n}^{\mathsf{O}(1)}$. It suffices to show that this implies that $\Pi_2^p \subseteq \Sigma_2^p$. In particular we use the fact there any NP problem has a polynomial-time circuit to find a Σ_2^p algorithm for the Π_2^p-complete problem $\Pi_2 SAT$. Recall that $\langle\varphi\rangle \in \Pi_2 SAT$ if and only if

$$\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n \; \varphi(u,v).$$

Observe that $\{(\langle\varphi\rangle, u) \mid \exists v \in \{0,1\}^n \, \varphi(u,v)\}$ is an NP language. Therefore, by assumption, there exists a circuit family $\{C_n\}_n$ of size $q(n)$ for some polynomial q such that $C_n(\langle\varphi\rangle, u) = 1$ if and only if $\exists v \in \{0,1\}^n \, \varphi(u,v)$. It would be then seem natural to define a Σ_2^p algorithm to existentially quantify over the bits of the encoding of $\langle C_n\rangle$ and then universally quantify over u. The difficulty is that we don't know that the bits sequence actually is for the correct circuit C_n that actually solve the NP problem. However, by applying the standard polynomial self-reduction for NP problems we can convert the circuit family $\{C_n\}_n$ to a circuit family $\{C'_n\}_n$ that on input $(\langle\varphi\rangle, u)$ actually produces a $v' \in \{0,1\}^n$ such that $\varphi(u,v')$ is true if one exists. (The circuit C'_n will have to make n calls to the circuit C_n successively fixing one bit of v' at a time so its size will be at most $nq(n)$ and thus $\langle C'_n\rangle$ will be at most $n^2q^2(n)$ bits long.) Therefore the Σ_2^p characterization of $\Pi_2 SAT$ is

$$\exists\langle C'_n\rangle \forall u \in \{0,1\}^n, \varphi(u, C'_n(\langle\varphi\rangle, u)),$$

which is what we needed to show $\square$

2.2 Probabilistic Complexity Classes

A *probabilistic (randomized) TM* is an ordinary multi-tape TM with an extra one-way read-only *coin flip (random)* tape. If the running time of M is $T(n)$ then on input x, the coin flip tape is initialized to a uniformly random string $r \in \{0,1\}^{f(|x|)}$ where $f(n) \leq T(n)$. If r is the string of coin flips for a machine M then we write $T(n)$ then $|r| \leq T(n)$. Now we can write $M(x,r)$ to denote the output of M on input x with random tape r where $M(x,r) = 1$ if M accepts and $M(x,r) = 0$ if M rejects.

A *probabilistic polynomial-time Turing Machine (PPT)* is a probabilistic Turing Machine whose worst-case running time $T(n)$ is polynomial in n.

We can now define several probabilistic complexity classes. (The terminology, due to Gill who introduced these classes, is not the most natural but it has stuck.)

Definition 2.5 Randomized Polynomial Time: $L \in \mathsf{RP}$ if and only if there exists a probabilistic polynomial time TM M such that for some error $\epsilon < 1$,

- $\forall w \in L$, $\Pr[M \text{ accepts } w] \geq 1 - \epsilon$, and
- $\forall w \notin L$, $\Pr[M \text{ accepts } w] = 0$.

equivalently $\forall w \in L$, $\Pr_r[M(w,r) = 1] \geq 1 - \epsilon$ and $\forall w \notin L$, $\Pr_r[M(w,r) = 1] = 0$.

The error, ϵ, is fixed for all input sizes. RP is the class of problems with one-sided error (i.e. an accept answer is always correct, whereas a reject may be incorrect.) coRP, which has one-sided error in the other direction, is defined analogously. The following class encompasses machines with two-sided error:

Definition 2.6 Bounded-error Probabilistic Polytime: $L \in \mathsf{BPP}$ if and only if there exists a probabilistic polynomial time TM M such that for some $\epsilon < \frac{1}{2}$,

- $\forall w \in L$, $\Pr[M \text{ accepts } w] \geq 1 - \epsilon$ and

- $\forall w \notin L$, $\Pr[M \text{ accepts } w] \leq \epsilon$.

If we identify the language L with its characteristic function $L(w) = \begin{cases} 1 & \text{if } w \in L \\ 0 & \text{if } w \notin L \end{cases}$ then we can write this equivalently as $L \in \mathsf{BPP}$ iff for all w we have $\Pr_r[M(w,r) = L(w)] \geq 1 - \epsilon$.

Clearly $\mathsf{RP} \subseteq \mathsf{NP}$, $\mathsf{coRP} \subseteq \mathsf{coNP}$. Also $\mathsf{RP}, \mathsf{coRP} \subseteq \mathsf{BPP}$ and BPP is closed under complement. Randomized algorithms with 1-sided or 2-sided errors such as these are known as *Monte Carlo* algorithms. Although we have so far required that the error in the definitions of BPP,RP, and coRP be constant we can consider the more general case when the error $\epsilon = \epsilon(n)$ is a function of the input size.

Definition 2.7 Zero-error Probabilistic Polytime: $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$.

Lemma 2.8 $L \in \mathsf{ZPP}$ if and only if there is a probabilistic TM M that always outputs the correct answer (i.e, $L(M) = L$) and the expected runtime of M is polynomial.

Proof
$\Rightarrow$: Let M_1 be an RP machine for L, and M_2 be a coRP machine for L with errors $\epsilon_1, \epsilon_2 < 1$. Define a probabilistic TN M that repeatedly runs M_1 followed by M_2 using independent random strings until one accepts. If either accepts then the answer must be correct so if M_1 accepts, then accept and if M_2 accepts then reject. Let $\epsilon = \max(\epsilon_1, \epsilon_2)$. We expect to have to run at most $\frac{1}{1-\epsilon}$ trials before one accepts. Thus M decides L in polynomial expected time.

$\Leftarrow$: Let $T(n)$ be the expected running time of a probabilistic TM M that always outputs the correct answer for language L. By Markov's inequality the probability that M runs for more than $3T(n)$ steps is at most $1/3$. To get an RP algorithm M' for L truncate the computation of M after $3T(n)$ steps. If M has accepted then accept, otherwise reject. If $w \in L$ then M' will accept w with probability at least $2/3$ and if $w \notin L$ then M will not accept w no matter what the random string. The algorithm for coRP is completely dual. $\square$

Randomized algorithms that are always correct but may run forever are known as Las Vegas algorithms. Our last probabilistic complexity class is much more powerful:

Definition 2.9 Probabilistic Polytime: $L \in \mathsf{PP}$ if and only if

$$\Pr_r[M(w,r) = L(w)] > \frac{1}{2}.$$

Here the error is allowed to be exponentially close to $1/2$, which is the key difference from BPP.

Note that with PP, it might take exponentially many trials even to notice the probability advantage.

2.2.1 Amplification

Lemma 2.10 For any probabilistic TM M with running time $T(n)$ and two-sided error $\epsilon(n) = \frac{1}{2} - \delta(n)$ there is a probabilistic TM M' with running time at most $O(\frac{m}{\delta^2(n)}T(n))$ and error at most 2^{-m} for the same language.

Proof M' simply runs M some number, k, times and takes the majority vote. The result follows by simple Chernoff bounds. We give a detailed calculation below. The error is:

$$\begin{aligned}
\Pr_r[M'(x,r) \neq L(x)] &= \Pr[\geq \frac{k}{2} \text{ wrong answers on } x] \\
&= \sum_{i=0}^{k/2} Pr[\frac{k}{2} + i \text{ wrong answers of } M \text{ on } x] \\
&= \sum_{i=0}^{k/2} \binom{k}{\frac{k}{2}+i} \epsilon^{\frac{k}{2}+i}(1-\epsilon)^{\frac{k}{2}+i} \\
&\leq \sum_{i=0}^{k/2} \binom{k}{\frac{k}{2}+i} \epsilon^{\frac{k}{2}}(1-\epsilon)^{\frac{k}{2}} \\
&\leq 2^k \epsilon^{\frac{k}{2}}(1-\epsilon)^{\frac{k}{2}} \\
&= \left[4(\frac{1}{2}-\delta)(\frac{1}{2}+\delta)\right]^{\frac{k}{2}} \\
&= (1-4\delta^2)^{\frac{k}{2}} \\
&\leq e^{-2\delta^2 k} \quad \text{since } 1-x \leq e^{-x} \\
&\leq 2^{-m} \quad \text{for } k = \frac{m}{\delta^2}
\end{aligned}$$

□

Note that amplification from sub-constant to constant error allows us to generalize the definition of BPP to allow $\epsilon = \epsilon(n) = 1/2 - \delta(n)$ for $\delta(n) \geq 1/q(n)$ for any polynomial q. However for PP, this amplification does not yield an efficient algorithm since $\delta(n)$ may be 2^{-n}.

A similar approach can be used with an RP language, this time accepting if any of the k trials accept. This gives an error of ϵ^k, where we can choose $k = \frac{m}{\log(\frac{1}{\epsilon})}$.

2.3 Randomness and Non-uniformity

The following theorem show that randomness is no more powerful than advice in general.

Theorem 2.11 (Gill, Adleman) BPP $\subseteq$ P/poly.

Proof Let $L \in$ BPP. By the amplification lemma, there exists a BPP machine M for L and a polynomial p such that:

$$\forall x \Pr_{r \in \{0,1\}^{p(n)}}[M(x,r) \neq L(x)] \leq 2^{-n-1}.$$

For $r \in \{0,1\}^{p(n)}$ say that r is *bad for x* iff $M(x,r) \neq L(x)$. By assumption, for all $x \in \{0,1\}^n$,

$$\Pr_r[r \text{ is bad for } x] \leq 2^{-n-1}$$

We say that r is *bad* if there exists an $x \in \{0,1\}^n$ such that r is bad for x.

$$\begin{aligned}\Pr_r[r \text{ is bad}] &\leq \sum_{x\in\{0,1\}^n} \Pr_r[r \text{ is bad for } x] \\ &\leq 2^n 2^{-n-1} \leq 1/2 < 1.\end{aligned}$$

Therefore for every n there must exist an $r_n \in \{0,1\}^{p(n)}$ such that r_n is not bad. (In fact this is true by construction for at least half the strings in $\{0,1\}^{p(n)}$.) We can use this sequence $\{r_n\}_n$ as the advice sequence for a $\mathsf{P/poly}$ machine that decides L. Each advice string is a particular random string r_n that leads to a correct answer for every input of length n. $\square$

2.4 BPP and the Polynomial-time Hierarchy

We know that $\mathsf{RP} \subseteq \mathsf{NP}$. Here we see that generalizing to bounded 2-sided error still stays within PH.

Theorem 2.12 (Sipser-Gacs, Lautemann) $\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Proof Note that BPP is closed under complement, so it suffices to show $\mathsf{BPP} \subseteq \Sigma_2^p$.
Let $L \in \mathsf{BPP}$. Then by amplification, there is a probabilistic polytime TM M and polynomial $p(n)$ such that

$$\Pr_{r\in\{0,1\}^{p(n)}}[M(x,r) \neq L(x)] \leq 2^{-n}.$$

Define $Acc_M(x) = \{r \in \{0,1\}^{p(n)} \mid M(x,r) = 1\}$. We have two cases: either $Acc_M(x)$ is almost all of $\{0,1\}^{p(n)}$ and we should accept x or $Acc_M(x)$ is only an exponentially small fraction of $\{0,1\}^{p(n)}$ and we should reject x. Moreover, we have a polynomial-time algorithm to determine membership in $Acc_M(x)$, namely on input r simply run $M(x,r)$.

The general property we will prove is that for if a set $S \subseteq \{0,1\}^m$ contains a large fraction of $\{0,1\}^m$ then a small number of translations of S will cover $\{0,1\}^m$ but if S is a small fraction of $\{0,1\}^m$ then no small set of translations will suffice to cover the set. The translation we use is just bit-wise exclusive or of bit vectors, $\oplus$. For $S \subseteq \{0,1\}^m$ and $t \in \{0,1\}^m$, define $S\oplus t = \{s\oplus t | s \in S\}$. Note that $|S \oplus t| = |S|$ and that $b \in S \oplus t$ if and only if $t \in S \oplus b$.

Lemma 2.13 (Lautemann) Let $S \subseteq \{0,1\}^m$. If $\frac{|S|}{2^m} > \frac{1}{2}$ then there exists $t_1, \ldots, t_m \in \{0,1\}^m$ such that,

$$\bigcup_{j=1}^{m} (S \oplus t_j) = \{0,1\}^m.$$

Proof By the probabilistic method.
Let $|S| > 2^{m-1}$ be a sufficiently large set as defined above. Choose $t_1, \cdots, t_m$ uniformly and independently at random from $\{0,1\}^m$. Fix a string $b \in \{0,1\}^m$ and $j \in [m] = \{1, \ldots, m\}$.

$$\Pr[b \in S \oplus t_j] = \Pr[t_j \in S \oplus b] = \Pr[t_j \in S] > \frac{1}{2}.$$

Therefore for any $j \in [m]$, $\Pr[b \notin S \oplus t_j] < 1/2$. The probability that b is not in any of the m translations is then

$$\Pr[b \notin \bigcup_{j=1}^{m} (S \oplus t_j)] = \prod_{j=1}^{m} \Pr[b \notin S \oplus t_j] < 2^{-m}.$$

Therefore

$$\Pr[\exists b \in \{0,1\}^m \text{ s.t. } b \notin \bigcup_{j=1}^{m} (S \oplus t_j)] < 2^m 2^{-m} = 1.$$

Therefore there exists a set $t_1, \ldots, t_m$ such that the union of the translations of S by the t_j covers all strings in $\{0,1\}^m$. $\square$

Now apply Lautemann's lemma with $S = Acc_M(x)$ and $m = p(n)$. If $x \notin L$ then $Acc_M(x)$ is only a 2^{-n} fraction of $\{0,1\}^m$, and so m translations will only be able to cover at most an $p(n)2^{-n}$ fraction of $\{0,1\}^m$, certainly not all of it. This gives us the following Σ_2^p characterization of L:

$$x \in L \Leftrightarrow \exists (t_1, \ldots, t_{p(|x|)}) \in \{0,1\}^{p^2(|x|)} \forall r \in \{0,1\}^{p(|x|)} (M(x, r \oplus t_1) = 1 \vee \ldots \vee M(x, r \oplus t_{p(|x|)}) = 1).$$

$\square$

Lecture 3

Circuit Size versus Uniform Complexity

April 8, 2008
Lecturer: Paul Beame
Notes:

The Karp-Lipton theorem gives a conditional result about the relationship between circuit complexity (non-uniform complexity) and uniform complexity. What unconditional properties do we know about circuit complexity and about its relationship to uniform complexity?

Let $\mathbb{B}_n = \{f : \{0,1\}^n \to \{0,1\}\}$, that is, the set of all Boolean functions on n bits. Observe that $|\mathbb{B}_n| = 2^{2^n}$.

Theorem 3.1 (Shannon) "Most" Boolean functions $f : \{0,1\}^n \to \{0,1\}$, have circuit complexity $size(f) \geq \frac{2^n}{n} - \phi(n)$ where $\phi(n)$ is $o(\frac{2^n}{n})$. More precisely, for any $\epsilon > 0$ and any basis $\Omega \subseteq \mathbb{B}_1 \cup \mathbb{B}_2$ there is a function $\phi_\epsilon : \mathbb{N} \to \mathbb{N}$ such that at least a $(1-\epsilon)$ fraction of functions f have $size_\Omega(f) \geq \frac{2^n}{n} - \phi_\epsilon(n)$.

Proof The proof is a by a counting argument. We will show that the number of circuits of size much smaller than $\frac{2^n}{n}$ is only a negligible fraction of $|\mathbb{B}_n|$, proving the claim.

We first compute the number of circuits of with $S \geq n+2$ gates over n inputs with $\Omega = \{\neg, \wedge, \vee\}$. What does it take to specify a given circuit? A gate labeled i in the circuit is defined by the labels of its two inputs, j and k ($j = k$ for unary gates), and the operation g the gate performs. The input labels j and k can be any of the S gates or the n inputs or the two constants, 0 and 1. The operation g can be any one of the three Boolean operations in the basis $\{\neg, \wedge, \vee\}$. Therefore there are at most $(S+n+2)^2 3$ possibilities for each gate. The circuit description also needs to specify the output gate, so any circuit with at most S gates can be specified by a description of length at most $(S+n+2)^{2S}3^S S$ (where we have added dummy gates if the circuit has fewer than S gates).

Note, however, that such descriptions compute the same function under each of the $S!$ ways of naming the gates. Since $S! \geq (S/e)^S$ for any integer S, the number of different functions computed by circuits of size S is at most

$$\begin{aligned}
\frac{(S+n+2)^{2S}3^S S}{S!} &\leq \frac{(S+n+2)^{2S}(3e)^S}{S^S} \\
&\leq \frac{(2S)^{2S}(3e)^S S}{S^S} \qquad \text{since } S \geq n+2 \\
&\leq (12eS)^{S+1}
\end{aligned}$$

Observe that for general $\Omega \subseteq \mathbb{B}_1 \cup \mathbb{B}_2$, we can assume that $|\Omega| \leq 16$ since constant functions and the unary identity function are not needed and we can replace 3 in the above calculation by 16. Therefore the number of such circuits is at most $(64eS)^{S+1}$. If $(64eS)^{S+1} \leq \epsilon 2^{2^n}$ then at least an $(1-\epsilon)$ fraction of functions in $\mathbb{B}_n$ have at size at least S. This holds if $(S+1)\log_2(64eS) \le 2^n - \log_2(1/\epsilon)$. Now for $S \le 2^n/n$, we have $\log_2(64eS) \le n+8-\log_2 n$ and so $(S+1)\log_2(64eS) \le (S+1)(n+8-\log_2 n)$. Therefore if $S+1 \le \frac{2^n}{n+8}$ then $(S+1)\log_2(64eS) \le 2^n - \frac{2^n \log_2 n}{n+8}) \le 2^n - \log_2(1/\epsilon)$ for n sufficiently large as a function of $1/\epsilon$. The claim follows by observing that $\frac{1}{n+8} = \frac{1}{n} - \frac{8}{n(n+8)}$ which is $\frac{1}{n} - \Theta(\frac{1}{n^2})$. □

Theorem 3.2 (Lupanov) Every Boolean function $f : \{0,1\}^n \to \{0,1\}$ has $size(f) \le \frac{2^n}{n} + \psi(n)$ where $\psi(n)$ is $o(\frac{2^n}{n})$.

Proof Proof of this part is left as an exercise. Note that a Boolean function f over n variables can be easily computed in using its canonical DNF or CNF representation and so $size(f) \le n2^{n+1}$. Bringing it down close to $\frac{2^n}{n}$ is a bit trickier. This gives a fairly tight bound on the size needed to compute most Boolean functions over n variables. □

As a corollary, we get a circuit size hierarchy theorem which is even stronger than the time and space hierarchies we saw earlier; circuits can compute many more functions even when their size is tripled.

Corollary 3.3 (Circuit-size Hierarchy). For any $S, S' : \mathbb{N} \to \mathbb{N}$, if $n \le 3S(n) \le S'(n) < 2^n/n$, then $\mathsf{SIZE}(S(n)) \subsetneq \mathsf{SIZE}(S'(n))$.

Proof Let $m = m(n) < n$ be the largest integer such that $S'(n) \ge 2^m/m + \psi(m)$ where $\psi(m)$ is defined as in Theorem 3.2 and is $o(2^m/m)$. Since $2^{m+1}/(m+1) + \psi(m+1) > S'(n) \ge 3S(n)$, we have $S(n) < \frac{1}{3}(\frac{2^{m+1}}{m+1} + \psi(m+1)) = \frac{2}{3}(\frac{2^m}{m+1} + \frac{\psi(m+1)}{2}) = \frac{2}{3}(\frac{2^m}{m} - \frac{2^m}{m(m+1)} + \frac{\psi(m+1)}{2}) < \frac{2^m}{m} - \phi(m)$ where $\phi(m)$ is defined as in Theorem 3.1 for $\epsilon = 1/2$. Consider the set F of all Boolean functions on n variables that depend only on the first m bits of their inputs. By Theorem 3.2, all functions in F can be computed by circuits of size $2^m/m + \psi(m) \le S'(n)$ and are therefore in $\mathsf{SIZE}(S'(n))$. On the other hand, at least 1/2 of the functions in F cannot be computed by circuits of size $2^m/m - \phi(m) > S(n)$ and are therefore not in $\mathsf{SIZE}(S(n))$. (Note that with the weaker bound size upper bound based on DNF formulas of $m2^{m+1}$ for m-input functions, a similar argument would yield a separation if $S'(n)$ is $\omega(S(n)\log^2 S(n))$.) □

We now consider how these circuit size classes relate to uniform complexity classes. The Cook-Levin Theorem shows how to simulate any algorithm running $\mathsf{TIME}(T(n))$ on inputs of length n by a circuit of size $O(T^2(n))$ with a constant number of circuit elements for each entry of the $T(n) \times T(n)$ tableau for the time $T(n)$ computation. The following Theorem, whose proof we just sketch, shows that a more efficient simulation is possible.

Theorem 3.4 (Fischer-Pippenger) If $T(n) \ge n$ then $\mathsf{TIME}(T(n)) \subseteq \bigcup_c \mathsf{SIZE}(cT(n)\log_2 T(n))$.

Proof The basic idea of the proof is a variant on the Cook-Levin tableau construction. Observe that the only calculations that take place in each row of this tableau involve the constant number of circuit elements that surround the read/write head. The contents of other entries can just be passed along directly to the next row. Unfortunately, in a typical Turing machine running on inputs of size n the position of the read/write head at a fixed time step can vary based on the input string. We say that a multitape Turing machine is *oblivious* if and only if the positions of its read/write heads only depends on the time step but not on its actual input.

It turns out that Hennie and Stearns [1] showed that there is a simulation of multitape TMs running in time $O(T(n))$ by 2-tape oblivious TMs running in time $O(T(n)\log T(n))$. The circuit we need to construct is just the tabeau circuit for this 2-tape TM. This circuit will have two rows for each time step and only a constant number of circuit elements per row (after the first row). The total number of gates will be $O(T(N)\log T(n))$. $\square$

Theorem 3.5 (Kannan) For all k, $\Sigma_2^p \cap \Pi_2^p \not\subseteq \mathsf{SIZE}(n^k)$.

Proof We know that $\mathsf{SIZE}(n^{k+1}) \not\subseteq \mathsf{SIZE}(n^k)$ by the circuit hierarchy theorem. To prove this theorem we will give a specific example of a language with circuit size at least n^{k+1} that is in $\Sigma_2^p \cap \Pi_2^p \setminus \mathsf{SIZE}(n^k)$.

For each n, let C_n be the lexicographically smallest circuit on n inputs such that $size(C_n) \geq n^{k+1}$ and C_n is minimal; i.e., C_n is not equivalent to any smaller circuit. (For lexicographic ordering on circuit encodings, we'll use $\preceq$ and we assume that if $size(C) \leq size(C')$ then $C \preceq C'$.) Let $\{C_n\}_{n=0}^{\infty}$ be the corresponding circuit family and let A be the language decided by this family.

By our choice of C_n, $A \notin \mathsf{SIZE}(n^k)$. Also, by the circuit hierarchy theorem, $size(C_n)$ is a polynomial $\leq 3n^{k+1}$ and the size of the encoding $|\langle C_n\rangle| \leq n^{k+3}$, say. Note that the factor of 3 is necessary because there may not be a circuit of size exactly n^{k+1} that computes A, but there must be one of size not too much larger than this by the circuit hierarchy theorem. We first show a weaker reswlt.

CLAIM: $A \in \Sigma_4^p$.

The basic idea of the claim is that we can express the conditions using quantifiers. We define A by guessing the encoding $\langle C_n\rangle$ for inputs of length n as a string and then verifying that C_n satisfies:

- $size(C_n) \geq n^{k+1}$.
- C_n is minimal.
- For all minimal circuits D on n inputs of size at least n^{k+1} (and at most $3n^{k+1}$), $C_n \preceq D$.

Recall from Lecture 1 that the property of a circuit being minimal is a Π_2^p property. That is, a circuit C on n inputs (of size at most $3n^{k+1}$ say) is minimal if and only if

$$\forall \langle C'\rangle \in \{0,1\}^{n^{k+3}} \exists y \in \{0,1\}^n ((size(C') \geq size(C)) \vee (C'(y) \neq C(y))).$$

The third condition for a fixed D of size between n^{k+1} and $3n^{k+1}$ is equivalent to saying that D is not minimal or $C_n \preceq D$, i.e., . This is a Π_2^p condition in $\langle D\rangle$ and $\langle C_n\rangle$:

$$\exists \langle D'\rangle \in \{0,1\}^{n^{k+3}} \forall z \in \{0,1\}^n \ [((size(D') \leq size(D)) \wedge (D'(z) = D(z))) \vee (C_n \preceq D)].$$

Now, we can use the same variable D to represent the candidate circuit in the third condition and in place of the C' in the minimality condition for C_n. Therefore $x \in A$ if and only if

$$\begin{aligned}\exists\langle C\rangle \in \{0,1\}^{|x|^{k+3}}\,\forall\langle D\rangle \in \{0,1\}^{|x|^{k+3}}\,\exists\langle D'\rangle \in \{0,1\}^{|x|^{k+3}}\,\exists y \in \{0,1\}^{|x|}\,\forall z \in \{0,1\}^{|x|}&\\ (C(x)&\\ \wedge\, (size(C) \geq |x|^{k+1})&\\ \wedge\, ((size(D) \geq size(C)) \vee (D(y) \neq C(y)))&\\ \wedge\, [(size(D') \geq |x|^{k+1}) \wedge [((size(D) \leq size(D')) \wedge (D(z) = D'(z))) \vee (C \preceq D)]).&\end{aligned}$$

The last three lines of the condition each match an item of the requirements and specifies that the circuit C is precisely $C_{|x|}$. The first line says that $x \in A$ if and only if $C_{|x|}(x)$ is true. This proves the claim and also the weaker conclusion that $A \in \mathsf{PH}$.

We finish the proof of the theorem by analyzing two possible scenarios:

(a) $\mathsf{NP} \subseteq \mathsf{P/poly}$. In this case, by the Karp-Lipton Theorem, $A \in \mathsf{PH} = \Sigma_2^p \cap \Pi_2^p$ because the polynomial time hierarchy collapses, and we are done.

(b) $\mathsf{NP} \nsubseteq \mathsf{P/poly}$. In this simpler case, there is some $B \in \mathsf{NP} - \mathsf{P/poly}$. In particular $B \notin \mathsf{SIZE}(n^k)$ and since $\mathsf{NP} \subseteq \Sigma_2^p \cap \Pi_2^p$, we have $B \in \Sigma_2^p \cap \Pi_2^p - \mathsf{SIZE}(n^k)$.

This finishes the proof of the Theorem. $\square$

Note that this argument is non-constructive: A is an explicit language not in $\mathsf{SIZE}(n^k)$ and if $\mathsf{NP} \subseteq \mathsf{P/poly}$ then A is in $\Sigma_2^p \cap \Pi_2^p$. In the second case we do not have an explicit language B and we also don't explicitly know which case is true. The latter problem would not be an issue: We could define a new language $A \otimes B = \{0x \mid x \in A\} \cup \{1x \mid x \in B\}$, which is at least as hard as both A and B. However, the former problem is much trickier to deal with but we can get an explcit Σ_2^p (or Π_2) language that is not in $\mathsf{SIZE}(n^k)$.

The key to producing an explicit language in $\Sigma_2^p - \mathsf{SIZE}(n^k)$ is the fact that the proof of the Karp-Lipton theorem is constructive. The construction for the Karp-Lipton theorem shows that for any Π_2^p language L defined by an explicit formula $\forall u \in \{0,1\}^{q(|x|)} \exists v \in \{0,1\}^{q(|x|)} R(x,u,v)$ and for any polynomial circuit size bound n^k, there is another explicit formula

$$\exists\langle C'\rangle \in \{0,1\}^{n^{2k}} \forall u \in \{0,1\}^{q(|x|)} R(x,u,C'(x,u))$$

such that if the language L' defined by $\exists v \in \{0,1\}^{q(|x|)} R(x,u,v)$ is in $\mathsf{SIZE}(n^k)$ then the two formulas define the same language. In general the new formula might not define the same language so call this resulting Σ_2^p language $\tau(L)$; this will equal L if $L' \in \mathsf{SIZE}(n^k)$. Similarly, by taking complements, for $\bar{L} \in \Sigma_2^p$ there is an explicit $\tau(\bar{L}) \in \Pi_2^p$ that is equal to $\bar{L}$ if $L' \in \mathsf{SIZE}(n^k)$.

Now let $\bar{L}$ be the Σ_2^p language defined by removing the two initial quantifiers $\exists\langle C\rangle \in \{0,1\}^{|x|^{k+3}} \forall\langle D\rangle \in \{0,1\}^{|x|^{k+3}}$ from the definition of A. Then the Π_2^p language $\tau(\bar{L})$ is equal to $\bar{L}$ if $L' \in \mathsf{SIZE}(n^k)$ where L' is the NP language related to L defined as above. Now define A_0' by $x \in A_0'$ if and only if $\exists\langle C\rangle \in \{0,1\}^{|x|^{k+3}} \forall\langle D\rangle \in \{0,1\}^{|x|^{k+3}} x \in \tau(\bar{L})$. Clearly A_0' is an explicit Σ_3^p language and if $L' \in \mathsf{SIZE}(n^k)$ then $A_0' = A \notin \mathsf{SIZE}(n^k)$. Let $A' = A_0' \otimes L'$. Then A' is in Σ_3^p but $A' \notin \mathsf{SIZE}(n^k)$.

Repeating this construction again with A' instead of A and removing only the initial $\exists\langle C\rangle \in \{0,1\}^{|x|^{k+3}}$ from the Σ_3^p definition of A' we can apply the analogous transformation to the resulting Π_2^p language and convert A' to a $A'' = A_0'' \otimes L''$ that is in Σ_2^p but not in $\mathsf{SIZE}(n^k)$.

References

[1] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape Turing machines. *Journal of the ACM*, 13(4):533–546, 1966.

Lecture 4

Complexity classes of functions, #P, and the Permanent

April 10, 2008
Lecturer: Paul Beame
Notes:

We will now define some complexity classes of functions on input strings that output numerical values of strings rather than decision problems where the output is a single bit.

Definition 4.1 The class FP is the set of all functions $f : \{0,1\}^* \to \{0,1\}^*$ (or alternatively, $f : \{0,1\}^* \to \mathbb{N}$) that are computable by a TM with a separate output tape in polynomial time. The class FNP is the analogous class for nondeterministic polynomial time where the requirement for a nondeterministic TM to compute a function is that on a given input x the content of the output tape is $f(x)$ in all accepting computations.

Among the algorithmic problems representable by such functions are "counting problems" related to common decision problems. For example,

Definition 4.2 Let #3-*SAT* be the problem that takes as input the encoding $\langle\varphi\rangle$ of a 3-CNF formula φ and outputs the number of satisfying assignments of φ.

More generally, given an NP language A defined by $x \in A \Leftrightarrow \exists y \in \{0,1\}^{q(|x|)} R(x,y)$ for a natural polynomial-time computable predicate R associated with A, $\#A$ will be the function mapping x to $\#\{y \in \{0,1\}^{q(|x|)} \mid R(x,y)\}$. Note that this is not a precisely specified since there may be many different choices of R that will work for the same language A. For many natural problems, however, the right choice of the relation R will be obvious.

Definition 4.3 Given a complexity class C, we define the complexity class #C to be the set of all functions $f : \{0,1\}^* \to \mathbb{N}$ such that there is an $R \in$ C and a polynomial q such that $f(x) = \#\{y \in \{0,1\}^{q(|x|)} \mid (x,y) \in R\}$.

In particular, #P is the set of all functions $f : \{0,1\}^* \to \mathbb{N}$ such that there is an $R \in$ P and a polynomial q such that $f(x) = \#\{y \in \{0,1\}^{q(|x|)} \mid (x,y) \in R\}$.

4.1 Function classes and oracles

Let us consider the use of oracle TMs in the context of function classes. For a language A, let FP^A be the set of functions $f : \{0,1\}^* \to \{0,1\}^*$ that can be computed in polynomial time by an oracle TM $M^?$ that has A as an oracle.

Similarly, we can define oracle TMs $M^?$ that allow functions as oracles rather than sets. In this case, rather than receiving the answer from the oracle by entering one of two states, the machine can receive a binary encoded version of the oracle answer on an oracle answer tape. Thus for functions $f : \{0,1\}^* \to \{0,1\}^*$ or $f : \{0,1\}^* \to \mathbb{N}$ and a complexity class C for which it makes sense to define oracle versions, we can define C^f. For a complexity class FC' of functions we can define $\mathsf{C}^{\mathsf{FC}'} = \bigcup_{f \in \mathsf{FC}'} \mathsf{C}^f$.

Note that although $\#\mathsf{P}$ is a class of functions it is very closely related to the unbounded-error class PP. In fact, their closure under polynomial-time Turing reductions is the same.

Theorem 4.4 $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$.

Proof There are two directions. One is easy. For $L \in \mathsf{P}^{\mathsf{PP}}$ there is some $A \in \mathsf{PP}$ such that $L \in \mathsf{P}^A$. Furthermore there is some polynomial-time Turing machine M and polynomial p such that $x \in A$ if and only if $\#\{r \in \{0,1\}^{p(|x|)} \mid M(x,r) = 1\} > 2^{p(|x|)} - 1$. The $\mathsf{P}^{\#\mathsf{P}}$ algorithm will simply replace each call A to a call to the function $f \in \#\mathsf{P}$ that computes $\#\{r \in \{0,1\}^{p(|x|)} \mid M(x,r) = 1\}$ and then compares $f(x)$ to $2^{p(|x|)} - 1$.

For the other direction, one has to use a polynomial number of calls to an appropriate PP oracle to replace a call to the $\#\mathsf{P}$ oracle f that on input x returns $f(x) = \#\{y \in \{0,1\}^m \mid M(x,y) = 1\}$ where $m = q(|X|)$ for some polynomial-time computable M. We can denote M' be the TM that on input (z,y) where $y, z \in \{0,1\}^m$ accepts if $y \prec z$ in the lexicographic order. Define M that on input (x,z) flips $m+1$ bits by where $b \in \{0,1\}$. If $b = 0$ then run $M(x,y)$. If $b = 1$ then run $M'(z,y)$. The probability of acceptance of M'' is $(N_z + f(x))/2^{m+1}$ where z is the binary representation of integer $N_z \in \{0, \ldots, 2^m - 1\}$. If this is strictly larger than $1/2$ then we know that $f(x) > 2^m - N_z$. We can query this language with $m+1$ different values of z to do a binary search for the value of $f(z)$. □

Definition 4.5 A function f is $\#\mathsf{P}$-complete iff

1. $f \in \#\mathsf{P}$.
2. For all $g \in \#\mathsf{P}$ we have $g \in \mathsf{FP}^f$.

As 3-SAT is NP-complete, #3-SAT is $\#\mathsf{P}$-complete:

Theorem 4.6 #3-SAT is $\#\mathsf{P}$-complete.

Proof The reduction produced by the Cook-Levin tableau is "parsimonious", in that it preserves the number of solutions. More precisely, in circuit form there is precisely one satisfying assignment for the circuit for each NP witness y. Moreover, the conversion of the circuit to 3-SAT enforces precisely one satisfying assignment for each of the extension variables associated with each gate. □

Since the standard reductions are frequently parsimonious, they can be used to prove $\#\mathsf{P}$-completeness of many counting problems relating to NP-complete problems. In some instances they are not parsimonious but can be made parsimonious. For example we have the following.

Theorem 4.7 $\#HAM\text{-}CYCLE$ is #P-complete.

The set of #P-complete problems is not restricted to the counting versions of NP-complete problems, however; interestingly, problems in P can have #P-complete counting problems as well.

Consider $\#CYCLE$, the problem of finding the number of directed simple cycles in a graph G. (The corresponding problem $CYCLE$ is in P).

Theorem 4.8 $\#CYCLE$ is #P-complete.

Proof We reduce from $\#HAM\text{-}CYCLE$. We will map the input graph G for $\#HAM\text{-}CYCLE$ to a graph G' for $\#CYCLE$. Say G has n vertices. G' will have a copy u' of each vertex $u \in V(G)$, and for each edge $(u, v) \in E(G)$ the gadget in Figure 4.1 will be added between u' and v' in G'. This gadget consists of $N = n \lceil \log_2 n \rceil + 1$ layers of pairs of vertices, connected to u' and v' and connected by $4N$ edges within. The number of paths from u' to v' in G' is $2^N \geq 2n^n$. Each simple cycle of length ℓ in G yields $(2^N)^\ell = 2^{N\ell}$ simple cycles in G'. If G has k Hamiltonian cycles, there will be $k2^{Nn}$ corresponding simple cycles in G'. Now G has feweer than n^n simple cycles of any length, in particular of length $\leq n - 1$. The total number of simple cycles in G' corresponding to these cycles of length $\leq n - 1$ is $< n^n 2^{N(n-1)} = 2^{Nn-1}$ since $n^n \leq 2^{N-1}$. Therefore we compute $\#HAM\text{-}CYCLE(G) = \lfloor \#CYCLE(G')/2^{Nn} \rfloor$. □

Figure 4.1: Edge replacement in $\#HAM\text{-}CYCLE$ to $\#CYCLE$ reduction.

The following corollary is left as an exercise:

Corollary 4.9 $\#2\text{-}SAT$ is #P-complete.

4.2 Determinant and Permanent

Some interesting problems in matrix algebra Given an $n \times n$ matrix $A = (a_{ij})$, the determinant of A is

$$det(A) = \sum_{\sigma \in S_n} (-1)^{sgn(\sigma)} \prod_{i=1}^{n} a_{i\sigma(i)},$$

where S_n is the set of permutations of $[n] = \{1, \ldots, n\}$ and $sgn(\sigma)$ is the is the number of transpositions required to produce σ modulo 2. This problem is in FP.

The $(-1)^{sgn(\sigma)}$ is apparently a complicating factor in the definition of $det(A)$, but if we remove it we will see that the problem actually becomes harder. This is called the *permanent* of matrix A:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i\sigma(i)}.$$

Let $PERM$ be the problem of computing the permanent of a matrix. and 0-1$PERM$ the problem in the case that the matrix has binary entries. We can view the matrix A as the weighted adjacency matrix of a bipartite graph on $[n] \times [n]$. Each $\sigma \in S_n$ corresponds to a perfect matching in this graph. If we view the weight of a matching as the product of the weights of its edges the permanent is the total weight of all matchings in the graph.

In particular a 0-1 matrix A corresponds to an unweighted bipartite graph G for which A is the adjacency matrix, and $perm(A)$ represents the number of perfect matchings on G. Let $\#BIPARTITE\text{-}MATCHING$ be the problem of counting all such matchings. Therefore 0-1$PERM = \#BIPARTITE\text{-}MATCHING \in \#\mathsf{P}$.

Alternatively, an $n \times n$ matrix A can be viewed as a weighted adjacency matrix of a directed graph G on n vertices (with possibly self-loops). Now each permutation $\sigma \in S_n$ can be decomposed into a union of disjoint cycles. For example, if $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 2 & 1 & 6 \end{pmatrix} \in S_6$ then σ can also be written in cycle form as $(1\ 3\ 5)(2\ 4)(6)$ where the notation implies that each number in the group maps to the next and the last maps to the first. These cycles cover all of the points $[n]$. For a directed graph G, a *cycle-cover of* G is a union of simple cycles of G that contains each vertex precisely once. In particular the edges corresponding to a term $\prod_{i=1}^{n} a_{i\sigma(i)}$ is the permanent of A corresponds to the product of the weights of edges in the directed graph G corresponding to the cycle-cover corresponding to σ, which we can view as the weight of the cycle-cover. Therefore $perm(A)$ is the total weight of all cycle-covers of G.

For a weighted, directed graph G, define $PERM(G)$ as the total weight of all cycle-covers of G, where the weight of a cycle-cover is the product of the weights of all its edges. Thus, for an unweighted graph G, $PERM(G)$ is the number of cycle-covers of G. The hardness of 0-1$PERM$ is established by showing that the problem of finding the number of cycle-covers of G is hard.

Theorem 4.10 (Valiant) 0-1$PERM$ is $\#\mathsf{P}$-complete.

Proof We will reduce $\#3\text{-}SAT$ to 0-1$PERM$ in two steps. Given any 3-SAT formula ϕ, in the first step, we will create a weighted directed graph G' (with small weights) such that

$$PERM(G') = 4^{3m}\#(\varphi)$$

where m is the number of clauses in φ. In second step, we will convert G' to an unweighted graph G such that $PERM(G') = PERM(G) \mod M$, where M will only have polynomially many bits.

First, we will construct G' from ϕ. The construction will be via gadgets. The VARIABLE gadget is shown in Figure 4.2. All the edges have unit weights. Notice that it contains one dotted edge for every occurrence of the variable in ϕ. Each dotted edge will be replaced by a subgraph which will be described later. Any cycle-cover either contains all dotted edges of positive occurrence (and all self-loops of negative occurrence) or vice versa.

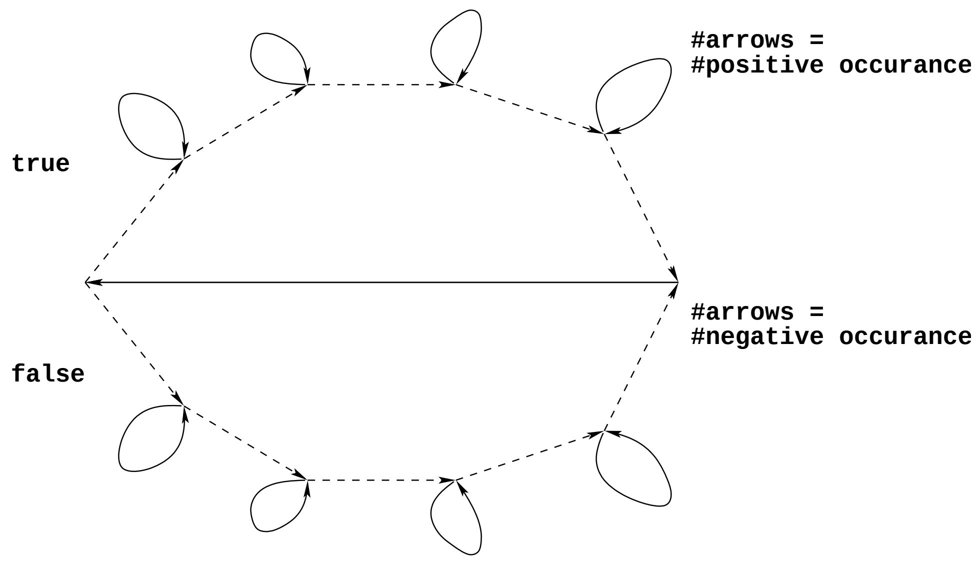

Figure 4.2: The VARIABLE gadget

The CLAUSE gadget is shown in Figure 4.2. It contains three dotted edges corresponding to three variables that occur in that clause. All the edges have unit weights. This gadget has the property that in any cycle-cover, at least one of the dotted edges is not used.

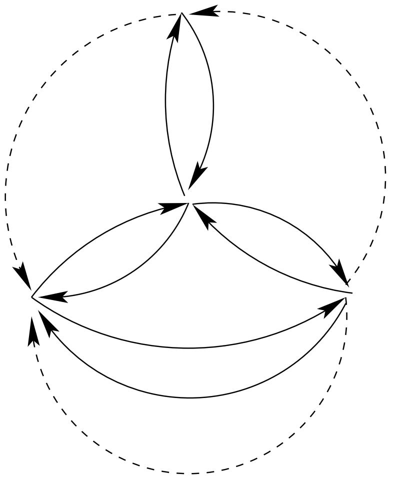

Figure 4.3: The CLAUSE gadget

Now, given any clause C and any variable x contained in it, there is a dotted edge (u, u') in the CLAUSE gadget for the variable and a dotted edge (v, v') in the VARIABLE gadget for the clause. These two dotted edges are replaced by an XOR gadget shown in Figure 4.2.

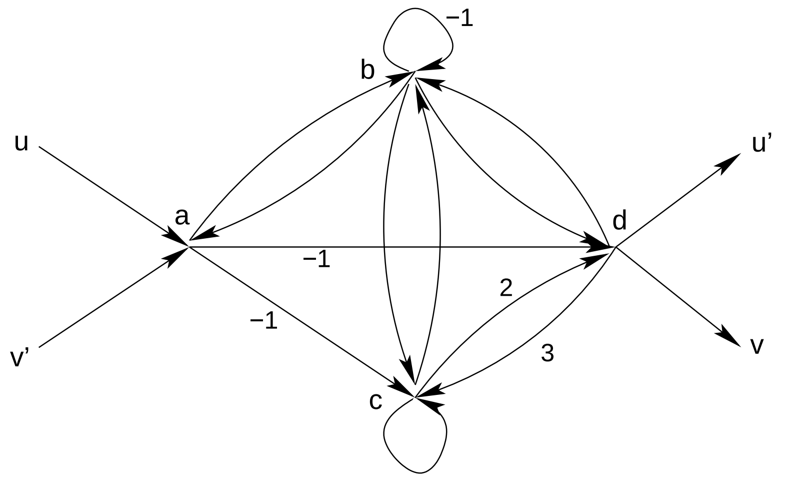

Figure 4.4: The CLAUSE gadget

The XOR gadget has the property that the total contribution of all cycle-covers using none or both of (u, u') and (v, v') is 0. For cycle-covers using exactly one of the two, the gadget contributes a factor of 4. To see this, lets consider all possibilities:

1. None of the external edges are present: The cycle-covers are [a c b d], [a b][c d], [a d b][c] and [a d c b]. The net contribution is (-2) + 6+ (-1) + (-3) = 0.
2. (u, a) and (a, v') are present: The cycle-covers are [b c d], [b d c], [c d][b] and [c][b d]. The net contribution is (2) + (3)+ (-6) + (1) = 0.
3. (v, d) and (d, u') are present: The cycle-covers are [a b][c] and [a c b]. The net contribution is 1 + (-1) = 0.
4. All four external edges are present: The cycle-covers are [b c] and [b][c]. The net contribution is 1 + (-1) = 0.
5. (v, d) and (a, v') are present: The cycle-covers are [d b a][c] and [d c b a]. The net contribution is 1 + 3 = 4.
6. (u, a) and (d, v') are present: The cycle-covers are [a d][b c], [a d][b][c], [a b d][c], [a c d][b], [a b c d] and [a c b d]. The net contribution is (-1) + 1 + 1 + 2 + 2 + (-1) = 4.

There are $3m$ XOR gadgets. As a result, every satisfying assignment of truth values to ϕ will contribute 4^{3m} to the cycle-cover and every other assignment will contribute 0. Hence,

$$PERM(G') = 4^{3m}\#(\phi)$$

Now, we will convert G' to an unweighted graph G. Observe that $PERM(G') \leq 4^{3m}2^n \leq 2^{6m+n}$. Let $N = 6m + n$ and $M = 2^N + 1$. Replace the weighted edges in G' with a set of unweighted edges as shown in Figure 4.2. For weights 2 and 3, the conversion does not affect the total weight of cycle-covers. For weight -1, the conversion blows up the total weight by $2^N \equiv -1 (mod M)$. As a result, if G is the resulting unweighted graph, PERM(G') = PERM(G) mod M.

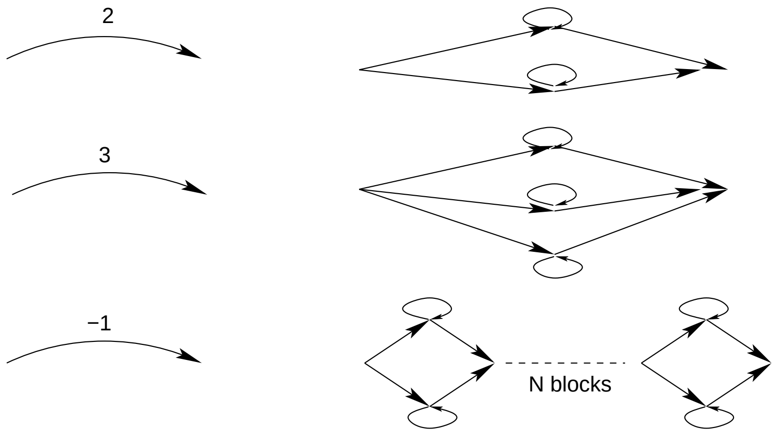

Figure 4.5: The VARIABLE gadget

Thus, we have shown a reduction of #3-SAT to 0-1PERM. This proves the theorem.

□