# Chapter 8

# Complexity of counting

In an **NP** decision problem, we ask the question of whether there *exists* a solution (a.k.a. certificate/witness) to a given instance. However, in many contexts, one is interested to know not only if a solution exists, but also the *number* of such solutions. This chapter studies #**P**, a complexity class that captures this notion.

Counting problems arise in diverse fields, such as statistical estimation, statistical physics, network design, etc. Counting problems are also studied in a field of mathematics called *enumerative combinatorics*, which tries to obtain closed-form mathematical expressions for counting problems. To give an example, the number of *spanning trees* in a graph can be counted by means of a simple determinant computation. Results in this chapter will show that for many natural counting problems, such efficiently computable expressions are unlikely to exist.

Here is an example that suggests how counting problems can arise in situations having to do with estimations of probability.

EXAMPLE 8.1 In the Graph − Reliability problem we are given a directed graph on $n$ nodes. Suppose we are told that each node can fail with probability 1/2 and told to compute the probability that node 1 has a path to $n$.

A moment's thought shows that under this simple edge failure model, the remaining graph is uniformly chosen at random from all subgraphs of the original graph. Thus the correct answer is

$$\frac{1}{2^n}(\text{number of subgraphs in which node 1 has a path to } n.)$$

We can view this as a *counting* version of the PATH problem.

In the rest of the chapter, we will study the class #**P**, that contains the Graph − Reliability problem and many other interesting counting problems. We also show a surprising connection between **PH** and #**P**, called *Toda's Theorem*. Along the way we encounter related classes such as $\oplus$**P**.

## 8.1 The class #P

We now define the class #**P**, which can be viewed as the counting version of **NP**. Up until now, we mostly focused our attention on *decision problems*, where for any given input, the output is a YES/NO (or 1/0) answer. However, the natural way to define a counting problem is that for any given input, the output is a number in **N** (representing the number of solutions). Thus, we define #**P** as a subset of the functions from $\{0,1\}^*$ to **N**.

DEFINITION 8.2 (#**P**) #**P** is the set of all functions $f : \{0,1\}^* \to \mathbf{N}$ such that there is a polynomial time NDTM $M$ such that for all $x \in \{0,1\}^*$,

$f(x) =$ number of accepting branches in $M$'s computation graph on $x$

As in the case of **NP**, we can define #**P** in an equivalent way, as follows:

DEFINITION 8.3 (#**P**, ALTERNATIVE DEFINITION) For $R \subseteq \{0,1\}^* \times \{0,1\}^*$ a relation, define $\#R$ to be a function from $\{0,1\}^*$ to **N** such that for every $x \in \{0,1\}^*$,

$$\#R(x) = |\{y : \ (x,y) \in R\}|$$

We say that $f \in \#\mathbf{P}$ if $f = \#R$ for a relation $R$ that is polynomial-time verifiable. That is, there is a polynomial time (deterministic) TM $M$ and a polynomial $p : \mathbf{N} \to \mathbf{N}$ satisfying:

1. For every $(x,y) \in R$, $|y| \leq p(|x|)$
2. For every $x, y \in \{0,1\}^*$, $M(x,y) = 1$ iff $(x,y) \in R$.

REMARK 8.4 Similar to the case of search problems, even when studying counting complexity, we can often restrict our attention to *decision problems* in the sense that there we can define a class of decision problems **PP** such that

$$\mathbf{PP} = \mathbf{P} \Leftrightarrow \#\mathbf{P} = \mathbf{FP} \tag{1}$$

The class **PP** corresponds to the most significant bit of #**P**. That is, $L$ is in **PP** if there exists a polynomial-time TM $M$ and a polynomial $p : \mathbf{N} \to \mathbf{N}$ such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \left| \left\{ y \in \{0,1\}^{p(|x|)} : M(x,y) = 1 \right\} \right| \geq \frac{1}{2} \cdot 2^{p(|x|)}$$

. You're asked to prove the non-trivial direction of (1) in Exercise 1. It is instructive to compare the class **PP**, which we believe contains problem requiring exponential time to solve, with the class **BPP**, which although has a seemingly similar definition, can in fact be solved efficiently using probabilistic algorithms (and perhaps even also using deterministic algorithms, see Chapter 19). Note that we do not know whether the class of decision problems corresponding to the *least* significant bit of #**P**, namely $\oplus$**P**, is also equivalent in its power to #**P**.

We will be interested in understanding which #**P**-counting problems are in fact in **FP** (recall that **FP** is the analog of the class **P** for functions, that is, **FP** is the set of functions computable by a deterministic polynomial-time Turing machine).

Here are two examples for problems in #**P**:

- #SAT is the problem of computing, given a boolean function $\phi$, the number of satisfying assignments for $\phi$.

- #CYCLE is the problem of computing, given a directed graph $G$, the number of simple cycles in $G$. (A simple cycle is one that does not visit any node twice.)

Clearly, if #SAT $\in$ **FP** then SAT $\in$ **P** and so **P** = **NP**. Thus presumably #SAT $\notin$ **FP**. How about #CYCLE? The corresponding decision problem —given a directed graph decide if it has a cycle—can be solved in linear time by breadth-first-search. The next theorem suggests that the counting problem may be much harder.

THEOREM 8.5
*If* #CYCLE $\in$ **FP**, *then* **P** = **NP**.

PROOF: We show that if #CYCLE can be computed in polynomial time, then Ham $\in$ **P**, where Ham is the **NP**-complete problem of deciding whether or not a given digraph has a Hamiltonian cycle. Given a graph $G$ with $n$ nodes in the Ham problem, we construct a graph $G'$ for #CYCLE such that $G$ has a Ham iff $G'$ has at least $n^{n^2}$ cycles.

To obtain $G'$, replace each edge $(u, v)$ in $G$ by a gadget as shown in Figure **??**. The gadget has $N = n \log_2 n + 1$ levels. It is an acyclic digraph, so cycles in $G'$ correspond to cycles in $G$. Furthermore, there are $2^N$ directed paths from $u$ to $v$ in the gadget, so a simple cycle of length $l$ in $G$ yields $(2^N)^l$ simple cycles in $G'$.

Figure unavailable in pdf file.

Figure 8.1: Reducing Ham to #CYCLE

Notice, if $G$ has a Hamiltonian cycle, then $G'$ has at least $(2^N)^n > n^{n^2}$ cycles. If $G$ has no Hamiltonian cycle, then the longest cycle in $G$ has length at most $n-1$. The number of cycles is bounded above by $n^{n-1}$. So $G'$ can have at most $(2^N)^{n-1} \times n^{n-1} < n^{n^2}$ cycles.

□

## 8.2 #P completeness.

Now we define #**P**-completeness. Loosely speaking, a function $f$ is #**P**-complete if it is in #**P** and if a polynomial-time algorithm for $f$ will imply that #**P** = **FP**. To formally define #**P**-completeness, we will use the notion of *oracle* TMs, defined in Chapter 4. Recall that according to the notation used there, $\mathbf{FP}^f$ is the set of functions that are computable by polynomial time TMs that have access to an oracle for function $f$.

Definition 8.6 A function Function $f$ is is #**P**-*complete* if it is in #**P** and every $g \in \#\mathbf{P}$ is also in $\mathbf{FP}^f$

If $f \in \mathbf{FP}$ then $\mathbf{FP}^f = \mathbf{FP}$. Thus the following is immediate.

Proposition 8.7
*If $f$ is ♯-complete and $f \in \mathbf{FP}$ then $\mathbf{P} = \mathbf{NP}$.*

Counting versions of many **NP**-complete languages such as 3SAT,Ham, CLIQUE naturally lead to #**P**-complete problems. The reason is that the standard reductions used to show that these languages are **NP**-hard preserve the number of certificates (such a reductions are called *parsimonious*). For example, the Cook-Levin reduction from an **NP** language $L$ decided by an NDTM $M$ to 3SAT (see Section **??**) transformed any input $x$ into a formula $\varphi$ such that the number of satisfying assignments to $\varphi$ is exactly equal to the number of accepting computation branches of the machine $M$ on the input $x$, implying:

Theorem 8.8
#SAT *is* #**P**-*complete*

However, as we'll see below, there are #**P**-complete problems for which the corresponding decision problems are in fact in **P**.

### 8.2.1 Permanent and Valiant's Theorem

Now we study another problem. The *permanent* of an $n \times n$ matrix $A$ is defined as

$$\mathsf{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i\sigma(i)} \tag{2}$$

where $S_n$ denotes the set of all permutations of $n$ elements. Recall that the expression for the determinant is similar

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

except for an additional "sign" term.[1] This similarity does not translate into computational equivalence: the determinant can be computed in polynomial time, whereas computing the permanent seems much harder, as we see below.

The permanent function can also be interpreted combinatorially. First, suppose the matrix $A$ has each entry in $\{0,1\}$. It may be viewed as the adjacency matrix of a bipartite graph $G(X,Y,E)$, with $X = \{x_1,\ldots,x_n\}$, $Y = \{y_1,\ldots,y_n\}$ and $\{x_i,y_j\} \in E$ iff $A_{ij} = 1$. Then the term $\prod_{i=1}^{n} a_{i\sigma(i)}$ is 1 iff $\sigma$ is a *perfect matching* (which is a set of $n$ edges such that every node is in exactly one edge). Thus $\mathsf{perm}(A)$ is simply the number of perfect matchings in $G$. Thus $\mathsf{perm}(\cdot) \in \#\mathbf{P}$ for $0,1$ matrices. If $A$ is a $\{-1,0,1\}$ matrix, then $\mathsf{perm}(A) = \left|\left\{\sigma : \prod_{i=1}^{n} a_{i\sigma(i)} = 1\right\}\right| - \left|\left\{\sigma : \prod_{i=1}^{n} a_{i\sigma(i)} = -1\right\}\right|$, so one can make two calls to a #SAT oracle to compute $\mathsf{perm}(A)$. Thus $\mathsf{perm}(\cdot) \in \mathbf{FP}^{\#\mathsf{SAT}}$ in this case. In fact one can show for general integer matrices that computing the permanent is in $\mathbf{FP}^{\#\mathsf{SAT}}$.

The next theorem came as a surprise to researchers in the 1970s, since it implies that if $\mathsf{perm} \in \mathbf{FP}$ then $\mathbf{P} = \mathbf{NP}$. Thus the permanent may be computationally much more difficult then the determinant.

THEOREM 8.9 (VALIANT)
perm *for* $0,1$ *matrices is* #**P***-complete.*

[1] It is known that every permutation $\sigma \in S_n$ can be represented as a composition of transpositions, where a transposition is a permutation that only switches between two elements in $[n]$ and leaves the other elements intact (one proof for this statement is the Bubblesort algorithm). If $\tau_1,\ldots,\tau_m$ is a sequence of transpositions such that their composition equals $\sigma$, then the *sign* of $\sigma$ is equal to $+1$ if $m$ is even and $-1$ if $m$ is odd. It can be shown that the sign is well-defined in the sense that it does not depend on the representation of $\sigma$ as a composition of transpositions.

Figure unavailable in pdf file.

Figure 8.2: This graph has permanent 0

Before proving Theorem 8.9, we introduce yet another way to look at the permanent. Consider matrix $A$ as the the adjacency matrix of a weighted $n$-node digraph (with possible self loops). Then the expression $\prod_{i=1}^{n} a_{i\sigma(i)}$ is nonzero iff $\sigma$ is a cycle-cover of $A$ (A *cycle cover* is a subgraph in which each node has in-degree and out-degree 1; such a subgraph must be composed of cycles.) We define the *weight* of the cycle cover to be the product of the weights of the edges in it. As a warm-up for the proof, consider the following example:

EXAMPLE 8.10 Consider the graph in Figure 8.2. (Unmarked edges have unit weight. We follow this convention through out this chapter.) Even without knowing what the subgraph $G'$ is, we show that the permanent of the whole graph is 0. For each cycle cover in $G'$ there are exactly two cycle covers for the three nodes, one with weight 1 and one with weight $-1$. Any non-zero weight cycle cover of the whole graph is composed of a cycle cover for $G'$ and one of these two cycle covers. Thus the sum of the weights of all cycle covers of $G$ is 0.

Now we prove Valiant's Theorem.

PROOF: We shall reduce the #**P**-complete problem #3SAT to perm. Given a boolean formula $\phi$ with $n$ variables and $m$ clauses, first we shall show how to construct an integer matrix $A'$ with negative entries such that $\mathsf{perm}(A') = 4^m \cdot (\#\phi)$. ($\#\phi$ stands for the number of satisfying assignments of $\phi$). Later we shall show how to to get a 0-1 matrix $A$ from $A'$ such that knowing $\mathsf{perm}(A)$ allows us to compute $\mathsf{perm}(A')$.

The main idea is that there are two kinds of cycle covers in the digraph $G'$ associated with $A'$: those that correspond to satisfying assignments (we will make this precise) and those that don't. Recall that $\mathsf{perm}(A')$ is the sum of weights of all cycle covers of the associated digraph, where the weight of a cycle cover is the product of all edge weights in it. Since $A'$ has negative entries, some of these cycle covers may have negative weight. Cycle covers of negative weight are crucial in the reduction, since they help cancel out contributions from cycle covers that do not correspond to satisfying assignments. (The reasoning to prove this will be similar to that in Example **??**.) On the other hand, each satisfying assignment contributes $4^m$ to $\mathsf{perm}(A')$, so $\mathsf{perm}(A') = 4^m \cdot (\#\phi)$.

Figure unavailable in pdf file.

Figure 8.3: The Variable-gadget

Figure unavailable in pdf file.

Figure 8.4: The Clause-gadget

To construct $G'$ from $\phi$, we use three kinds of gadgets as shown in Figures **??**, **??** and **??**. There is a variable gadget per variable and a clause gadget per clause. There are two possible cycle covers of a variable gadget, corresponding to an assignment of 0 or 1 to that variable. Assigning 1 corresponds to a single cycle taking all the external edges ("true-edges"), and assigning 0 correspond to taking all the self-loops and taking the "false-edge". Each external edge of a variable is associated with a clause in which the variable appears.

The clause gadget is such that the only possible cycle covers exclude at least one external edge. Also for a given (proper) subset of external edges used there is a unique cycle cover (of weight 1). Each external edge is associated with a variable appearing in the clause.

We will also use a graph called the XOR gadget (Figure **??**) which has the following purpose: we want to ensure that exactly one of the edges $uu'$ and $vv'$ (see the schematic representation in Figure **??**) is used in a cycle cover that contributes to the total count. So after inserting the gadget, we want to count only those cycle covers which either enter the gadget at $u$ and leave it at $u'$ or enter it at $v$ and leave it at $v'$. This is exactly what the gadget guarantees: one can check that the following cycle covers have total weight of 0: those that do not enter or leave the gadget; those that enter at $u$ and leave at $v'$, or those that enter at $v$ and leave at $u'$. In other words, the only cycle covers that have a nonzero contribution are those that either (a) enter at $u$ and leave at $u'$ (which we refer to as using "edge" $uu'$) or (b) enter at $v$ and leave at $v'$ (referred to as using edge $vv'$). These are cycle covers in the "schematic graph" (which has edges as shown in Figure **??**) which *respect* the XOR gadget.

The XOR gadgets are used to connect the variable gadgets to the corresponding clause gadgets so that only cycle covers corresponding to a satisfying assignment need be counted towards the total number of cycle covers. Consider a clause, and a variable appearing in it. Each has an external edge

Figure unavailable in pdf file.

Figure 8.5: The XOR-gadget

Figure unavailable in pdf file.

Figure 8.6: For each clause and variable appearing in it, an XOR-gadget connects the corresponding external edges. There are $3m$ such connections in total.

corresponding to the other, connected by an XOR gadget (figure **??**). If the external edge in the clause is not taken (and XOR is respected) the external edge in the variable must be taken (and the variable is true). Since at least one external edge of each clause gadget has to be omitted, each cycle cover respecting all the XOR gadgets corresponds to a satisfying assignment. (If the XOR is not respected, we need not count such a cycle cover as its weight will be cancelled by another cover, as we argued above). Conversely, for each satisfying assignment, there is a cycle cover (unique, in the schematic graph) which respects all the XOR gadgets.

Now, consider a satisfying assignment and the corresponding cycle cover in the schematic graph. Passing (exactly one of) the external edges through the XOR gadget multiplies the weight of each such cover by 4. Since there are $3m$ XOR gadgets, corresponding to each satisfying assignment there are cycle covers with a total weight of $4^{3m}$ (and all other cycle covers total to 0). So $\mathsf{perm}(G') = 4^{3m}\#\phi$.

Finally we have to reduce finding $\mathsf{perm}(G')$ to finding $\mathsf{perm}(G)$, where $G$ is an unweighted graph. First we reduce it to finding $\mathsf{perm}(G)$ modulo $2^N + 1$ for a large enough $N$ (but still polynomial in $|G'|$). For this, we can replace -1 edges with edges of weight $2^N$, which can be converted to $N$ edges of weight 2 in series. Changing edges of (small) positive integral weights (i.e., multiple or parallel edges) to unweighted edges is as follows: cut each (repeated) edge into two and insert a node to connect them; add a self loop to the node. This does not change the permanent, and the new graph has only unweighted edges. □

*details + figures*

### 8.2.2 Approximate solutions to #P problems

Since computing exact solutions to #**P**-complete problems is presumably difficult, one should try to compute approximate solutions in the sense of

the following definition.

DEFINITION 8.11 Let $f, g : \{0,1\}^* \to \mathbf{N}$ be functions and $c > 1$. We say that $f$ *approximates* $g$ *within a factor* $c$ if for every string $x$, $g(x) \leq f(x) \leq c \cdot g(x)$.

Not all #**P** problems behave identically with respect to this notion. Approximating certain problems within any constant factor $c$ is **NP**-hard (see Exercises). For other problems such as 0/1 permanent, there is a *Fully polynomial randomized approximation scheme* (FPRAS), which is an algorithm which, for any $\epsilon, \delta$, approximates the function within a factor $1+\epsilon$ (its answer may be incorrect with probability $\delta$) in time $\mathrm{poly}(n, \log 1/\delta, \log 1/\epsilon)$. This algorithm —as well as other similar algorithms for a host of #**P**-complete problems—use the *Monte Carlo Markov Chain* technique. The result that spurred this development is due to Valiant and Vazirani and it shows that under fairly general conditions, approximately counting the number of elements in a set ( membership in which is testable in polynomial time) is equivalent —in the sense that the problems are interreducible via polynomial-time randomized reductions— to the problem of generating a *random sample* from the set. We will not discuss this interesting area any further, though we will further explore the complexity of approximation in the exercises.

## 8.3 Toda's Theorem: $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathsf{SAT}}$

An important question in the 1980s was the relative power of the polynomial-hierarchy **PH** and the class of counting problems #**P**. Both are natural generalizations of **NP**, but it seemed that their features— alternation and the ability to count witnesses, respectively — are not directly comparable to each other. Thus it came as big surprise when in 1989 Toda showed:

THEOREM 8.12 (TODA'S THEOREM [TOD91])
$\mathbf{PH} \subseteq \mathbf{P}^{\#\mathsf{SAT}}$.

That is, we can solve any problem in the polynomial hierarchy given an oracle to a #**P**-complete problem. Now we prove this result, following the proof of [KVVY93].

### 8.3.1 The class $\oplus$**P** and hardness of satisfiability with unique solutions.

The following complexity class will be used in the proof:

DEFINITION 8.13 A language $L$ in the class $\oplus\mathbf{P}$ (pronounced "parity P") iff there exists a polynomial time NTM $M$ such that $x \in L$ iff the number of accepting paths of $M$ on input $x$ is odd.

Thus, $\oplus\mathbf{P}$ can be considered as the class of decision problems corresponding to the least significant bit of a $\#\mathbf{P}$-problem. As in the proof of Theorem 8.8, the fact that the standard **NP**-completeness reduction is parsimonious implies the following problem $\oplus\mathsf{SAT}$ is $\oplus\mathbf{P}$-complete (under many-to-one Karp reductions):

DEFINITION 8.14 Define the quantifier $\bigoplus$ as follows: for every Boolean formula $\varphi$ on $n$ variables. $\bigoplus_{x\in\{0,1\}^n} \varphi(x)$ is true if the number of $x$'s such that $\varphi(x)$ is true is odd.[2] The language $\oplus\mathsf{SAT}$ consists of all the true quantified Boolean formula of the form $\bigoplus_{x\in\{0,1\}^n} \varphi(x)$ where $\varphi$ is an unquantified Boolean formula (not necessarily in CNF form).

Unlike the class $\#\mathbf{P}$, it is not immediately clear that a polynomial-time algorithm for $\oplus\mathbf{P}$ implies that $\mathbf{NP} = \mathbf{P}$. However, we will show that such an algorithm will imply that $\mathbf{NP} = \mathbf{RP}$.

THEOREM 8.15
*There's a probabilistic polynomial-time algorithm $A$ such that for every $n$-variable Boolean formula $\varphi$*

$$\varphi \in \mathsf{SAT} \Rightarrow \Pr[A(\varphi) \in \oplus\mathsf{SAT}] \geq \tfrac{1}{4n}$$
$$\varphi \notin \mathsf{SAT} \Rightarrow \Pr[A(\varphi) \in \oplus\mathsf{SAT}] = 0$$

**Pairwise independent hash functions.**

To prove Theorem 8.15 we will use the notion of *pairwise independent hash functions*.

DEFINITION 8.16 (PAIRWISE INDEPENDENT HASH FUNCTIONS) Let $\mathcal{H}_{n,k}$ be a collection of functions from $\{0,1\}^n$ to $\{0,1\}^k$. We say that $\mathcal{H}_{n,k}$ is *pairwise independent* if for every $x, x' \in \{0,1\}^n$ with $x \neq x'$ and for every $y, y' \in \{0,1\}^k$,

$$\Pr_{h\in_R\mathcal{H}_{n,k}}[h(x) = y \ \wedge \ h(x') = y'] = 2^{-2n}$$

[2]Note that if we identify true with 1 and 0 with false then $\bigoplus_{x\in\{0,1\}^n} \varphi(x) = \sum_{x\in\{0,1\}^n} \varphi(x) \pmod 2$. Also note that $\bigoplus_{x\in\{0,1\}^n} \varphi(x) = \bigoplus_{x_1\in\{0,1\}} \cdots \bigoplus_{x_n\in\{0,1\}} \varphi(x_1, \ldots, x_n)$.

Note that an equivalent formulation is that for every two distinct strings $x, x' \in \{0,1\}^n$ the random variable $\langle h(x), h(x')\rangle$ for $h$ chosen at random from $\mathcal{H}_{n,k}$ is distributed according to the uniform distribution on $\{0,1\}^k \times \{0,1\}^k$. The following theorem provides a construction of an *efficiently computable* pairwise independent hash functions (see also Exercise 4 for a different construction).

THEOREM 8.17 (EFFICIENT PAIRWISE INDEPENDENT HASH FUNCTIONS)
*For every function $k : \mathbf{N} \to \mathbf{N}$ such that $k(n)$ is at most a polynomial in $n$ and is computable in poly$(n)$ time, there exists a two-input polynomial-time TM $H$ such that for every $n \in \mathbf{N}$, the collection of functions $\mathcal{H}_{n,k(n)} = \{x \mapsto H(h,x)\}_{h\in\{0,1\}^{2\max\{n,k(n)\}}}$ is a pairwise independent hash function collection.*

PROOF: We can assume without loss of generality that $n \geq k$ since we can transform a pairwise collection with input size $n' > n$ into a collection with input size $n$ by simply padding the input with zeros. We start by assuming $k = n$. Recall that we can identify the strings in $\{0,1\}^n$ with elements of the field GF$(2^n)$. That is, we have operations $+$ and $\cdot$ on pairs of strings in $\{0,1\}^n$ satisfying the usual laws of commutativity, associativity and the distributive law. There also exist additive and multiplicative units, which we denote by $\mathbf{0}$ and $\mathbf{1}$, such that for every $x \in \{0,1\}^n$, $x + \mathbf{0} = x$ and $x \cdot \mathbf{1} = x$, and every element $x$ has a unique additive inverse, denoted $-x$, and if $x \neq \mathbf{0}$ it also has a unique multiplicative inverse, denoted $x^{-1}$. Furthermore, all the operations of multiplication, addition and finding inverses can be done in polynomial-time.

The collection of hash functions will be very simple: for every $a, b \in \{0,1\}^n$, we define the function $h_{a,b} : \{0,1\}^n \to \{0,1\}^n$ as follows: $h_{a,b}(x) = a \cdot x + b$. It is clearly efficiently computable, and so we need to verify that it is a pairwise independent collection. For every $x \neq x' \in \{0,1\}^n$ and $y, y' \in \{0,1\}^n$ we have that $h_{a,b}(x) = y$ and $h_{a,b}(x') = y'$ iff $a, b$ satisfy the equations:

$$\begin{aligned} a \cdot x + b &= y \\ a \cdot x' + b &= y' \end{aligned}$$

which imply $a \cdot (x - x') = y - y'$ or $a = (y - y')(x - x')^{-1}$. Since $b = y - a \cdot x$, we get that the pair $\langle a, b\rangle$ is completely determined by these equations, and so the probability that this happens over the choice of $a, b$ is exactly one over the number of possible pairs, and indeed equals $\frac{1}{2^{2n}}$.

For the case $k < n$ note that if we take a hash function from $\{0,1\}^n$ to $\{0,1\}^n$ and truncate its output to the first $k$ bits it is still a pairwise independent hash function. This holds since the truncation of the uniform distribution on $n$ bits to $k$ bits is the uniform distribution on $k$ bits, and so for every $x \neq x'$, we will have that the random variable $\langle h(x), h(x')\rangle$ is the uniform distribution over $\{0,1\}^k \times \{0,1\}^k$. □

Pairwise independent hash functions have several useful properties that led to numerous applications in theoretical and applied computer science. In this section, we will use the following result:

LEMMA 8.18 (VALIANT-VAZIRANI LEMMA [**?**])
*Let $\mathcal{H}_{n,k}$ be a pairwise independent hash function collection from $\{0,1\}^n$ to $\{0,1\}^k$ and $S \subseteq \{0,1\}^n$ such that $2^{k-2} \leq |S| \leq 2^{k-1}$. Then,*

$$\Pr_{h \in_R \mathcal{H}_{n,k}}[\left|\left\{x \in S : h(x) = 0^k\right\}\right| = 1] \geq \tfrac{1}{8}$$

PROOF: For every $x \in S$, let $p = 2^{-k}$ be the probability that $h(x) = 0^k$ when $h \in_R \mathcal{H}_{n,k}$. Note that $\frac{1}{4} \leq |S|p \leq \frac{1}{2}$. Let the event $A_x$ be the event that $h(x) = 0^k$ and $h(x') \neq 0^k$ for every $x' \in S$ with $x' \neq x$. We want to show that $\Pr[A] = \Pr[\cup_{x \in S} A_x] \geq \frac{1}{8}$. Now the events $\{A_x\}$ are disjoint and so we have that $\Pr[A] = \sum_{x \in S} \Pr[A_x]$. Now, because of pairwise independence, given that $h(x) = 0^k$, for every $x' \neq x$ the probability that $h(x') = 0^k$ is $p$, and hence by the union bound the probability that there exists some $x' \neq x$ with $h(x') = 0^k$ is at most $|S|p$. We get that for every $x \in S$, $\Pr[A_x] \geq p(1 - |S|p)$ and hence $\Pr[A] \geq |S|p(1 - |S|p)$. Since $\frac{1}{4} \leq |S|p \leq \frac{1}{2}$ we get that $\Pr[A] \geq \frac{1}{8}$. □

### Proof of Theorem 8.15

We'll now use Lemma 8.18 to prove Theorem 8.15. Given a formula $\varphi$ on $n$ variables, our probabilistic algorithm will choose $k$ at random from $\{2, \ldots, n+1\}$ and a random hash function $h \in_R \mathcal{H}_{n,k}$ and construct the formula

$$\psi = \bigoplus_{x \in \{0,1\}^n} \varphi(x) \wedge (h(x) = 0^n)$$

(such a formula can be found using the Cook-Levin reduction, perhaps using some auxiliary variables.)

If $\varphi$ is unsatisfiable then $\psi$ is false, since we'll have no $x$'s satisfying the inner formula and zero is an even number. If $\varphi$ is satisfiable, we let $S$

be the set of its satisfying assignments. With probability $1/n$, $k$ satisfies $2^{k-2} \leq |S| \leq 2^k$, which implies that with probability $1/8$, we'll have that there is a unique $x$ such that $\varphi(x) \wedge h(x) = 0^n$. Since one happens to be an odd number, this implies that $\psi$ is true. □

Remark 8.19 (Hardness of Unique Satisfiability) Note that the proof of Theorem 8.15 actually implied a stronger statement: that the existence of an algorithm that can distinguish between an unsatisfiable Boolean formula and a formula with exactly one satisfying assignment implies the existence of a probabilistic polynomial-time algorithm for all of **NP**. Thus, the guarantee that a particular search problem has either no solutions or a unique solution does not necessarily make the problem easier to solve.

### 8.3.2 Step 1: Randomized reduction from PH to $\oplus$P

We now go beyond **NP** and show that we can actually reduce any language in the polynomial hierarchy to $\oplus\mathsf{SAT}$. That is, we prove the following lemma:

Lemma 8.20
*Let $c \in \mathbf{N}$ be some constant. There exists a probabilistic polynomial-time algorithm $A$ such that for every $\psi$ a Quantified Boolean formula with $c$ levels of alternations, it holds that*

$$\psi \text{ is true} \Rightarrow \Pr[A(\psi) \in \oplus\mathsf{SAT}] \geq \tfrac{2}{3}$$
$$\psi \text{ is false} \Rightarrow \Pr[A(\psi) \in \oplus\mathsf{SAT}] = 0$$

Proof: Let $\psi$ be a formula with $c$ levels of alternating $\exists/\forall$ quantifiers, possibly with an initial $\bigoplus$ quantifier. We'll transform $\psi$ in probabilistic polynomial-time to a formula $\psi'$ such that $\psi'$ has only $c-1$ levels of alternating $\exists/\forall$ quantifiers, an initial $\bigoplus$ quantifier, and such that if $\psi$ was false then $\psi'$ will be false, and if $\psi$ was true then $\psi'$ will be true with probability $1 - \frac{1}{10c}$. We will prove the lemma by repeating this step $c$ times.

For a (possibly partially quantified) formula $\varphi$ on $n$ variables let $\#(\varphi)$ denote the number of satisfying assignments of $\varphi$. Before proving the lemma let us note that given two (possibly partially quantified) formulas $\varphi, \psi$ on variables $x \in \{0,1\}^n, y \in \{0,1\}^m$ we can construct in polynomial-time an $n+m$ variable formula $\varphi \cdot \psi$ and a $\max\{n,m\}+1$-variable formula $\varphi + \psi$ such that $\#(\varphi \cdot \psi) = \#(\varphi)\#(\varphi)$ and $\#(\varphi + \psi) = \#(\varphi) + \#(\psi)$. Indeed, take $\varphi \cdot \psi(x,y) = \varphi(x) \wedge \varphi(y)$ and $\varphi + \psi(z) = \big((z_0 = 0) \wedge \varphi(z_1, \ldots, z_n)\big) \vee \big((z_0 = 1) \wedge \psi(z_1, \ldots, z_m)\big)$. For a formula $\varphi$, we'll use the notation $\varphi + 1$ to denote the formula $\varphi + \psi$ where $\psi$ is some canonical formula with a single satisfying

assignment. Since the product of numbers is even iff one of the numbers is even, and since adding one flips the parity of a number, we get that for every two formulas $\varphi, \psi$ as above

$$\left(\bigoplus_x \varphi(x)\right) \vee \left(\bigoplus_y \psi(y)\right) \Leftrightarrow \bigoplus_{x,y} (\varphi \cdot \psi)(x,y) \tag{3}$$

$$\neg \bigoplus_x \varphi(x) \Leftrightarrow \bigoplus_z (\varphi + 1)(z) \tag{4}$$

Now let $\psi$ be a formula of the form $\bigoplus_x Q^1_{y_1} \cdots Q^c_{y_c} \varphi(x, y_1, \ldots, y_c)$, where $Q^1, \ldots, Q^c \in \{\exists, \forall\}$ and $x, y_1, \ldots, y_c$ range over poly$(n)$-long strings whose lengths we denote by $m, m_1, \ldots, m_c$ respectively. We need to transform $\psi$ into a formula of the same form with at most $c-1$ quantifiers. Assume $Q^1 = \exists$. In this case for every fixed value $x \in \{0,1\}^m$, the proof of Theorem 8.15 implies that we have a way to sample a formula $\tau$ on variables $x \in \{0,1\}^m, y \in \{0,1\}^{m'}$ (for some $m'$ polynomial in $n$) such that if the partially quantified formula $\psi'_{\restriction x}(y_1) = Q^2_{y_2} \cdots Q^c_{y_c} \varphi(x, y_1, \ldots, y_c)$ is unsatisfiable (i.e., false for every setting of $y_1$) then $\tau(x,y)$ is false for every $y$, and if there's some $y_1$ such that $\psi'_{\restriction x}(y_1)$ is true then with probability $\frac{1}{8n}$, there exists a unique (and in particular an odd number of) $y$ such that $\tau(x,y)$ is true. If we repeat this procedure $k = 100cnm$ times, we get $k$ formulas $\tau_1, \ldots, \tau_k$ such that if, in the notations from above, we let $\tau' = (\tau_1 + 1) \cdots (\tau_k + 1) + 1$ be a formula taking the variables $x \in \{0,1\}^m$ and $z \in \{0,1\}^{m''}$ where $m''$ is polynomial in $n$, then for every $x \in \{0,1\}^m$, we get that

- If there's no a $y_1$ with $\psi'_{\restriction x}(y_1)$ true then $\tau'(x,z)$ is false for all $z \in \{0,1\}^{m''}$.
- If there's such a $y_1$ then with probability $1 - \frac{1}{10c2^m}$, there's an odd number of $z$'s such that $\tau'(x,z)$ is true.

By taking a union bound over all the $2^m$ possible $x$'s, we get that $\psi \Leftrightarrow \bigoplus_{x,z} \tau'(x,z)$, thus completing the proof for the case that the first quantifier is $\exists$. In the case the first quantifier is $\forall$ we use the identities $\forall x \varphi(x) \Leftrightarrow \neg\exists x \neg\varphi(x)$ and $\bigoplus_x \neg\varphi(x) \Leftrightarrow \bigoplus_z (\varphi + 1)(z)$ to transform it into an $\exists$ quantifier. □

### 8.3.3 Step 2: Making the reduction deterministic

To complete the proof of Theorem 8.12, we prove the following lemma:

LEMMA 8.21
*There is a (deterministic) polynomial-time transformation $T$ that, for every formula $\psi$ that is an input for $\oplus\mathsf{SAT}$ we have that $T(\psi, 1^m)$ is an unquantified Boolean formula such that*

$$\psi \in \oplus\mathsf{SAT} \Rightarrow \#(\varphi) = -1 \pmod{2^{m+1}}$$
$$\psi \notin \oplus\mathsf{SAT} \Rightarrow \#(\varphi) = 0 \pmod{2^{m+1}}$$

**Proof of Theorem 8.12 from Lemma 8.21.** Let $L \in PH$. We show that we can decide whether an input $x \in L$ by asking a single question to a $\#\mathsf{SAT}$ oracle. Lemma 8.20 implies that there's a a polynomial-time TM $M$ such that if $x \in L$ then $\Pr_{r \in_R \{0,1\}^m}[M(x,r) \in \oplus\mathsf{SAT}] \geq 2/3$ and if $x \notin L$ then $\Pr_{r \in_R \{0,1\}^m}[M(x,r) \in \oplus\mathsf{SAT}] = 0$, where $m$ is the (polynomial) number of random bits used by our reduction. For every string $r \in \{0,1\}^m$, denote by $\psi_r$ the formula $M(x,r)$, and let $f(r) = 0$ if $\psi_r \notin \oplus\mathsf{SAT}$ and $f(r) = -1$ otherwise. For every $r, y$ let $P(r,y)$ denote the predicate that is true iff the assignment $y$ satisfies the formula $T(\psi_r, 1^m)$ where $T$ is the transformation obtained by Lemma 8.21. Since this predicate is computable in polynomial-time, invoking the Cook-Levin transformation, we obtain a formula $\tau$ taking variables $r, y$ and auxiliary variables $z$ such that for every $r$ the number of pairs $y, z$ such that $\tau(r,y,z)$ is true equals $\#(\varphi_r)$. In particular we have that

$$\#(\tau) = \sum_{r \in \{0,1\}^m} f(r) \pmod{2^{m+1}}$$

And thus if $x \notin L$ then $f(r) = 0$ for all $r$ then we $\#(\tau) = 0 \pmod{2^{m+1}}$. If $x \in L$ then let $\ell$ denote the number of $r$'s such that $f(r) = -1$. We have that $\#(\tau) = -\ell \pmod{2^{m+1}}$ but $\frac{2}{3}2^m \leq \ell \leq 2^m$ and hence this number is not equal to $0 \pmod{2^{m+1}}$. Thus, by a single query to a $\#\mathsf{SAT}$ oracle we can determine whether or not $x \in L$, establishing Theorem 8.12.

**Proof of Lemma 8.21.** Given formulas $\varphi$,$\tau$ recall that we defined formulas $\varphi+\tau$ and $\varphi\cdot\tau$ satisfying $\#(\varphi+\tau) = \#(\varphi)+\#(\tau)$ and $\#(\varphi\cdot\tau) = \#(\varphi)\#(\tau)$, and note that these formulas are of size at most a constant factor larger than $\varphi, \tau$. Consider the formula $4\tau^3 + 3\tau^4$ (where $\tau^3$ for example is shorthand for $\tau \cdot (\tau \cdot \tau)$). One can easily check that

$$\#(\tau) = -1 \pmod{2^{2^i}} \Rightarrow \#(4\tau^3 + 3\tau^4) = -1 \pmod{2^{2^{i+1}}} \quad (5)$$
$$\#(\tau) = 0 \pmod{2^{2^i}} \Rightarrow \#(4\tau^3 + 3\tau^4) = 0 \pmod{2)^{2^{i+1}}} \quad (6)$$

Let $\psi_0 = \psi$ and $\psi_{i+1} = 4\psi_i^3 + 3\psi_i^4$. Let

$$\psi^* = \psi_{\lceil \log(m+1) \rceil}$$

Repeated use of equations (5), (6) shows that if $\#(\psi)$ is odd, then $\#(\psi^*) = -1 \pmod{2^{m+1}}$ and if $\#(\psi)$ is even, then $\#(\psi^*) = 0 \pmod{2^{m+1}}$. Also, the size of $\psi^*$ is only polynomially larger than size of $\psi$. □

## 8.4 Open Problems

- What is the exact power of $\oplus$SAT and $\#SAT$ ?
- What is the average case complexity of $n \times n$ permanent modulo small prime, say 3 or 5 ? Note that for a prime $p > n$, random self reducibility of permanent implies that if permanent is hard to compute on at least one input then it is hard to compute on $1 - O(p/n)$ fraction of inputs, i.e. hard to compute on average (see Theorem **??**).

## Exercises

§1 Let $f \in \#\mathbf{P}$. Show a polynomial-time algorithm to compute $f$ given access to an oracle for some language $L \in \mathbf{PP}$ (see Remark 8.4). (Hint: without loss of generality you can think that $f = \#$CSAT, the problem of computing the number of satisfying assignments for a given Boolean circuit $C$, and that you are given an oracle that tells you if a given $n$-variable circuit, has at least $2^{n-1}$ satisfying assignments or not. The main observation you can use is that if $C$ has at least $2^{n-1}$ satisfying assignments then it is possible to use the oracle to find a string $x$ such that $C$ has exactly $2^{n-1}$ satisfying assignments that are larger than $x$ in the natural lexicographic ordering of the strings in $\{0,1\}^n$.)

§2 Show that computing the permanent for matrices with integer entries is in $\mathbf{FP}^{\#\mathsf{SAT}}$.

§3 Prove Theorem **??**.

§4 Let $k \leq n$. Prove that the following family $\mathcal{H}_{n,k}$ is a collection of pairwise independent functions from $\{0,1\}^n$ to $\{0,1\}^k$: Identify $\{0,1\}$ with the field GF(2). For every $k \times n$ matrix $A$ with entries in GF(2), and $k$-length vector $b \in \mathrm{GF}(2)^n$, $\mathcal{H}_{n,k}$ contains the function $h_{A,b} : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^k$ defined as follows: $h_{A,\mathbf{b}}(x) = Ax + b$.

§5 Show that if there is a polynomial-time algorithm that approximates #CYCLE within a factor 2, then $\mathbf{P} = \mathbf{NP}$.

§6 Show that for every $g \in \#\mathbf{P}$ and every $\epsilon > 0$, there is a function in $FP^{\mathbf{\Sigma}_3^p}$ that approximates $g$ within a factor $1 + \epsilon$. (Hint: Use hashing and ideas similar to those in the proof of $\mathbf{BPP} \subseteq \mathbf{PH}$, where we also needed to estimate the size of a set of strings.) Thus assuming **PH** doesn't collapse to a finite level, approximation is easier than exact computation.

§7 Show that every for every language in $\mathbf{AC}^0$ there is a depth 3 circuit of $n^{\text{poly}(\log n)}$ size that decides it on $1 - 1/\text{poly}(n)$ fraction of inputs and looks as follows: it has a single $\oplus$ gate at the top and the other gates are $\vee, \wedge$ of fanin at most $\text{poly}(\log n)$.

## Chapter notes and history

The definition of #**P** as well as several interesting examples of #**P** problems appeared in Valiant's seminal paper [Val79b]. The #**P**-completeness of the permanent is from his other paper [Val79a]. Toda's Theorem is proved in [Tod91].

For an introduction to FPRAS's for computing approximations to many counting problems, see the relevant chapter in Vazirani [Vaz01] ( an excellent resource on approximation algorithms in general).

NEED TO LIST SOME #**P**-COMPLETE PROBLEMS FROM PHYSICS.