

---

# Computational Complexity: A Modern Approach

*Draft of a book: Dated January 2007*  
Comments welcome!

Sanjeev Arora and Boaz Barak  
Princeton University  
complexitybook@gmail.com

---

Not to be reproduced or distributed without the authors' permission

This is an Internet draft. Some chapters are more finished than others. References and attributions are very preliminary and we apologize in advance for any omissions (but hope you will nevertheless point them out to us).

Please send us bugs, typos, missing references or general comments to  
complexitybook@gmail.com — **Thank You!!**

DRAFT

DRAFT

## Chapter 9

# Complexity of counting

*“It is an empirical fact that for many combinatorial problems the detection of the existence of a solution is easy, yet no computationally efficient method is known for counting their number.... for a variety of problems this phenomenon can be explained.”*

L. Valiant 1979

The class **NP** captures the difficulty of finding *certificates*. However, in many contexts, one is interested not just in a single certificate, but actually counting the *number* of certificates. This chapter studies **#P**, (pronounced “sharp p”), a complexity class that captures this notion.

Counting problems arise in diverse fields, often in situations having to do with estimations of probability. Examples include statistical estimation, statistical physics, network design, and more. 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, in the 19th century Kirchoff showed how to count the number of *spanning trees* in a graph using 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 estimations of probability.

---

### EXAMPLE 9.1

In the **GraphReliability** problem we are given a directed graph on  $n$  nodes. Suppose we are told that each node can fail with probability  $1/2$  and want 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 study the complexity class #P, a class containing the GraphReliability problem and many other interesting counting problems. We will show that it has a natural and important complete problem, namely the problem of computing the *permanent* of a given matrix. We also show a surprising connection between PH and #P, called *Toda's Theorem*. Along the way we encounter related complexity classes such as PP and  $\oplus P$ .

## 9.1 The class #P

We now define the class #P. Note that it contains functions whose output is a natural number, and not just 0/1.

### DEFINITION 9.2 (#P)

A function  $f : \{0, 1\}^* \rightarrow \mathbb{N}$  is in #P if there exists a polynomial  $p : \mathbb{N} \rightarrow \mathbb{N}$  and a polynomial-time TM  $M$  such that for every  $x \in \{0, 1\}^*$ :

$$f(x) = \left| \left\{ y \in \{0, 1\}^{p(|x|)} : M(x, y) = 1 \right\} \right|.$$

### REMARK 9.3

As in the case of NP, we can also define #P using non-deterministic TMs. That is, #P consists of all functions  $f$  such that  $f(x)$  is equal to the number of paths from the initial configuration to an accepting configuration in the configuration graph  $G_{M,x}$  of a polynomial-time NDTM  $M$ .

The big open question regarding #P, is whether all problems in this class are efficiently solvable. In other words, whether #P = FP. (Recall that FP is the analog of the class P for functions with more than one bit of output, that is, FP is the set of functions from  $\{0, 1\}^*$  to  $\{0, 1\}^*$  computable by a deterministic polynomial-time Turing machine. Thinking of the output as the binary representation of an integer we can identify such functions with functions from  $\{0, 1\}^*$  to  $\mathbb{N}$ . Since computing the number of certificates is at least as hard as finding out whether a certificate exists, if #P = FP then NP = P. We do not know whether the other direction also holds: whether NP = P implies that #P = FP. We do know that if PSPACE = P then #P = FP, since counting the number of certificates can be done in polynomial space.

Here are two more examples for problems in #P:

- #SAT is the problem of computing, given a Boolean formula  $\phi$ , 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 vertex 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

DRAFT

cycle—can be solved in linear time by breadth-first-search. The next theorem suggests that the counting problem may be much harder.

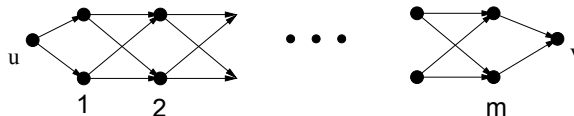


Figure 9.1: Reducing **Ham** to **#CYCLE**: by replacing every edge in  $G$  with the above gadget to obtain  $G'$ , every simple cycle of length  $\ell$  in  $G$  becomes  $(2^m)^\ell$  simple cycles in  $G'$ .

#### THEOREM 9.4

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 (i.e., a simple cycle that visits all the vertices in the graph). Given a graph  $G$  with  $n$  vertices, we construct a graph  $G'$  such that  $G$  has a Hamiltonian cycle iff  $G'$  has at least  $n^{n^2}$  cycles.

To obtain  $G'$ , replace each edge  $(u, v)$  in  $G$  by the gadget shown in Figure 9.1. The gadget has  $m = n \log n + 1$  levels. It is an acyclic digraph, so cycles in  $G'$  correspond to cycles in  $G$ . Furthermore, there are  $2^m$  directed paths from  $u$  to  $v$  in the gadget, so a simple cycle of length  $\ell$  in  $G$  yields  $(2^m)^\ell$  simple cycles in  $G'$ .

Notice, if  $G$  has a Hamiltonian cycle, then  $G'$  has at least  $(2^m)^n > n^{n^2}$  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  $(2^m)^{n-1} \times n^{n-1} < n^{n^2}$  cycles. ■

#### 9.1.1 The class **PP**: decision-problem analog for #P.

Similar to the case of search problems, even when studying counting complexity, we can often restrict our attention to *decision problems*. The reason is that there exists a class of decision problems **PP** such that

$$\mathbf{PP} = \mathbf{P} \Leftrightarrow \mathbf{\#P} = \mathbf{FP} \quad (1)$$

Intuitively, **PP** corresponds to computing the most significant bit of functions in **#P**. That is,  $L$  is in **PP** if there exists a polynomial-time TM  $M$  and a polynomial  $p : \mathbb{N} \rightarrow \mathbb{N}$  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 are 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 it has a seemingly similar definition, can in fact be solved efficiently using probabilistic algorithms (and perhaps even also using deterministic algorithms, see Chapter 16). Note that we do not know whether this holds also for the class of decision problems corresponding to the *least* significant bit of **#P**, namely  $\oplus\mathbf{P}$  (see Definition 9.13 below).

## 9.2 #P completeness.

Now we define #P-completeness. Loosely speaking, a function  $f$  is #P-complete if it is in #P and a polynomial-time algorithm for  $f$  implies that #P = FP. To formally define #P-completeness, we use the notion of *oracle* TMs, as defined in Section 3.5. Recall that a TM  $M$  has *oracle access* to a language  $O \subseteq \{0, 1\}^*$  if it can make queries of the form “Is  $q \in O$ ?” in one computational step. We generalize this to non-Boolean functions by saying that  $M$  has oracle access to a function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ , if it is given access to the language  $O = \{\langle x, i \rangle : f(x)_i = 1\}$ . We use the same notation for functions mapping  $\{0, 1\}^*$  to  $\mathbb{N}$ , identifying numbers with their binary representation as strings. For a function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ , we define  $\mathbf{FP}^f$  to be the set of functions that are computable by polynomial-time TMs that have access to an oracle for  $f$ .

DEFINITION 9.5

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

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

PROPOSITION 9.6

If  $f$  is #P-complete and  $f \in \mathbf{FP}$  then  $\mathbf{FP} = \mathbf{FP}^f$ .

Counting versions of many NP-complete languages such as 3SAT, Ham, and CLIQUE naturally lead to #P-complete problems. We demonstrate this with #SAT:

THEOREM 9.7

#SAT is #P-complete

PROOF: Consider the Cook-Levin reduction from any  $L$  in NP to SAT we saw in Section 2.3. This is a polynomial-time computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that for every  $x \in \{0, 1\}^*$ ,  $x \in L \Leftrightarrow f(x) \in \text{SAT}$ . However, the proof that the reduction works actually gave us more information than that. It provided a *Levin reduction*, by which we mean the proof showed a way to transform a *certificate* that  $x$  is in  $L$  into a certificate (i.e., satisfying assignment) showing that  $f(x) \in \text{SAT}$ , and also vice versa (transforming a satisfying assignment for  $f(x)$  into a witness that  $x \in L$ ).

In particular, it means that the mapping from the certificates of  $x$  to the assignments of  $f(x)$  was invertible and hence one-to-one. Thus the number of satisfying assignments for  $f(x)$  is equal to the number of certificates for  $x$ . ■

As shown below, there are #P-complete problems for which the corresponding decision problems are in fact in P.

### 9.2.1 Permanent and Valiant’s Theorem

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

$$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^n A_{i, \sigma(i)} \quad (2)$$

DRAFT

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

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

except for an additional “sign” term.<sup>1</sup> 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  $X = \{x_1, \dots, x_n\}$ ,  $Y = \{y_1, \dots, y_n\}$  and  $\{x_i, y_j\} \in E$  iff  $A_{i,j} = 1$ . 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 if  $A$  is a 0,1 matrix then  $\text{perm}(A)$  is simply the number of perfect matchings in the corresponding graph  $G$  and in particular computing  $\text{perm}(A)$  is in #P. If  $A$  is a  $\{-1, 0, 1\}$  matrix, then  $\text{perm}(A) = |\{\sigma : \prod_{i=1}^n A_{i\sigma(i)} = 1\}| - |\{\sigma : \prod_{i=1}^n A_{i\sigma(i)} = -1\}|$ , so one can make two calls to a #SAT oracle to compute  $\text{perm}(A)$ . In fact one can show for general integer matrices that computing the permanent is in  $\mathbf{FP}^{\#\text{SAT}}$  (see Exercise 2).

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

**THEOREM 9.8 (VALIANT’S THEOREM)**  
 $\text{perm}$  for 0,1 matrices is #P-complete.

Before proving Theorem 9.8, we introduce yet another way to look at the permanent. Consider matrix  $A$  as 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. Thus  $\text{perm}(A)$  is equal to the sum of weights of all possible cycle covers.

#### EXAMPLE 9.9

Consider the graph in Figure 9.2. Even without knowing what the subgraph  $G'$  is, we show that the permanent of the whole graph is 0. For each cycle cover in  $G'$  of weight  $w$  there are exactly two cycle covers for the three nodes, one with weight  $+w$  and one with weight  $-w$ . 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.

<sup>1</sup>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  $\tau_1, \dots, \tau_m$  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.

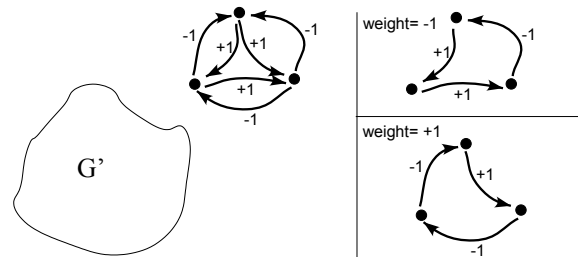


Figure 9.2: The above graph  $G$  has cycle cover weight zero regardless of the choice of  $G'$ , since for every cycle cover of weight  $w$  in  $G'$ , there exist two covers of weight  $+w$  and  $-w$  in the graph  $G$ . (Unmarked edges have  $+1$  weight; we follow this convention through out this chapter.)

**PROOF OF VALIANT'S THEOREM (THEOREM 9.8):** We 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  $\text{perm}(A') = 4^m \cdot (\#\phi)$ . ( $\#\phi$  stands for the number of satisfying assignments of  $\phi$ ). Later we shall show how to get a 0-1 matrix  $A$  from  $A'$  such that knowing  $\text{perm}(A)$  allows us to compute  $\text{perm}(A')$ .

The main idea is that our construction will result in 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. We will use negative weights to ensure that the contribution of the cycle covers that do not correspond to satisfying assignments cancels out. (This is similar reasoning to the one used in Example 9.9.) On the other hand, we will show that each satisfying assignment contributes  $4^m$  to  $\text{perm}(A')$ , and so  $\text{perm}(A') = 4^m \cdot (\#\phi)$ .

To construct  $G'$  from  $\phi$ , we combine the following three kinds of gadgets shown in Figure 9.3:

**Variable gadget** The variable gadget has two possible cycle covers, 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.

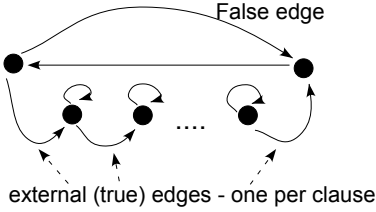
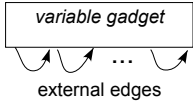
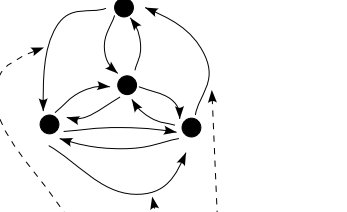
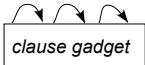
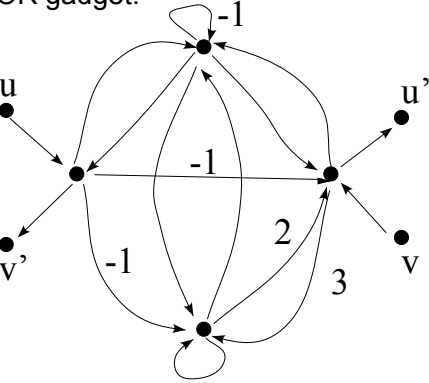
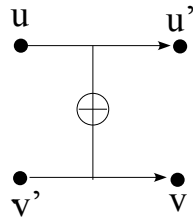
**Clause gadget** The clause gadget is such that the only possible cycle covers exclude at least one external edge. Also for a given (proper) subset of external edges used there is a unique cycle cover (of weight 1). Each external edge is associated with a variable appearing in the clause.

**XOR gadget** We also use a graph called the XOR gadget whose purpose is to ensure that for some pair of edges  $\overrightarrow{uu'}$  and  $\overrightarrow{vv'}$ , *exactly one* of these edges is present in any cycle cover that counts towards the final sum.

Suppose that we replace a pair of edges  $\overrightarrow{uu'}$  and  $\overrightarrow{vv'}$  in some graph  $G$  with the XOR gadget as described in Figure count:fig:valiantgad to obtain some graph  $G'$ . Then, via similar reasoning to Example 9.9, every cycle cover of  $G$  of weight  $w$  that uses exactly one of the edges  $\overrightarrow{uu'}$  and

DRAFT



Gadget:	Symbolic description:
<p>variable gadget:</p> 	
<p>clause gadget:</p> 	
<p>XOR gadget:</p> 	

The overall construction:

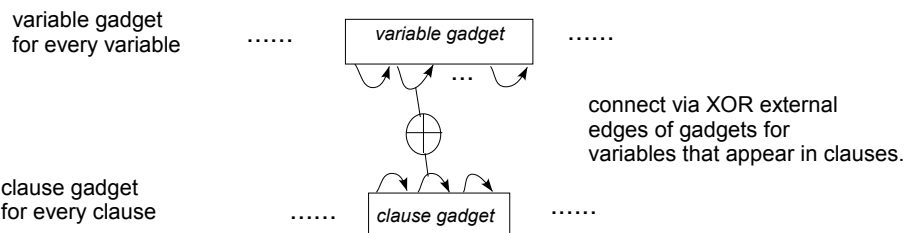


Figure 9.3: The gadgets used in the proof of Valiant's Theorem.

$\overrightarrow{vv'}$  is mapped to a set of cycle covers in  $G'$  whose total weight is  $4w$  (i.e., the set of covers that enter the gadget at  $u$  and exit at  $u'$  or enter it at  $v$  and exit it at  $v'$ ), while all the other cycle covers of  $G'$  have total weight 0 (Exercise 3). For this reason, whenever we replace edges  $\overrightarrow{uu'}$  and  $\overrightarrow{vv'}$  with a XOR gadget, we can consider in the analysis only cycle covers that use exactly one of these edges, as the other covers do not contribute anything to the total sum.

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 will be counted towards the total number of cycle covers. Consider a clause, and a variable appearing in it. Each has an external edge corresponding to the other, connected by an XOR gadget. If the external edge in the clause is not taken then by the analysis of the XOR gadget the external edge in the variable must be taken (and hence the variable is true). Since at least one external edge of each clause gadget has to be omitted, each cycle cover that is counted towards the sum corresponds to a satisfying assignment. Conversely, for each satisfying assignment, there is a set of cycle covers with total weight  $4^{3m}$  (since they pass through the XOR gadget exactly  $3m$  times). So  $\text{perm}(G') = 4^{3m} \# \phi$ .

**Reducing to the case 0,1 matrices.** Finally we have to reduce finding  $\text{perm}(G')$  to finding  $\text{perm}(G)$ , where  $G$  is an unweighted graph (or equivalently, its adjacency matrix has only 0,1 entries). We start by reducing to the case that all edges have weights in  $\{\pm 1\}$ . First, note that replacing an edge of weight  $k$  by  $k$  parallel edges of weight 1 does not change the permanent. Parallel edges are not allowed, but we can make edges non-parallel by cutting each edge  $\overrightarrow{uv}$  in two and inserting a new node  $w$  with an edge from  $u$  to  $w$ ,  $w$  to  $v$  and a self loop at  $w$ . To get rid of the negative weights, note that the permanent of an  $n$  vertex graph with edge weights in  $\{\pm 1\}$  is a number  $x$  in  $[-n!, +n!]$  and hence this permanent can be computed from  $y = x \pmod{2^{m+1}}$  where  $m$  is sufficiently large (e.g.,  $m = n^2$  will do). But to compute  $y$  it is enough to compute the permanent of the graph where all weight  $-1$  edges are replaced with edges of weight  $2^m$ . Such edges can be converted to  $m$  edges of weight 2 in series, which again can be transformed to parallel edges of weight  $+1$  as above. ■

### 9.2.2 Approximate solutions to #P problems

Since computing exact solutions to #P-complete problems is presumably difficult, a natural question is whether we can *approximate* the number of certificates in the sense of the following definition.

DEFINITION 9.10

Let  $f : \{0,1\}^* \rightarrow \mathbb{N}$  and  $\alpha < 1$ . An algorithm  $A$  is an  $\alpha$ -*approximation* for  $f$  if for every  $x$ ,  $\alpha f(x) \leq A(x) \leq f(x)/\alpha$ .

Not all #P problems behave identically with respect to this notion. Approximating certain problems within any constant factor  $\alpha > 0$  is NP-hard (see Exercise 5). 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  $\text{poly}(n, \log 1/\delta, \log 1/\epsilon)$ . Such approximation of counting problems is sufficient for many applications, in particular those where counting is needed to obtain

DRAFT

estimates for the probabilities of certain events (e.g., see our discussion of the graph reliability problem).

The approximation algorithm for the permanent —as well as other similar algorithms for a host of  $\#\mathbf{P}$ -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.

Interestingly, if  $\mathbf{P} = \mathbf{NP}$  then *every*  $\#\mathbf{P}$  problem has an FPRAS (and in fact an FPTAS: i.e., a *deterministic* polynomial-time approximation scheme), see Exercise 6.

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

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

THEOREM 9.11 (TODA'S THEOREM [?])  
 $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{SAT}}$ .

That is, we can solve any problem in the polynomial hierarchy given an oracle to a  $\#\mathbf{P}$ -complete problem.

#### REMARK 9.12

Note that we already know, even without Toda's theorem, that if  $\#\mathbf{P} = \mathbf{FP}$  then  $\mathbf{NP} = \mathbf{P}$  and so  $\mathbf{PH} = \mathbf{P}$ . However, this does not imply that any problem in  $\mathbf{PH}$  can be computed in polynomial-time using an oracle to  $\#\mathbf{SAT}$ . For example, one implication of Toda's theorem is that a *subexponential* (i.e.,  $2^{n^{o(1)}}$ -time) algorithm for  $\#\mathbf{SAT}$  will imply such an algorithm for any problem in  $\mathbf{PH}$ . Such an implication is not known to hold from a  $2^{n^{o(1)}}$ -time algorithm for  $\mathbf{SAT}$ .

#### 9.3.1 The class $\oplus\mathbf{P}$ and hardness of satisfiability with unique solutions.

The following complexity class will be used in the proof:

##### DEFINITION 9.13

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 9.7, the fact that the standard  $\mathbf{NP}$ -completeness reduction is parsimonious implies the following problem  $\oplus\mathbf{SAT}$  is  $\oplus\mathbf{P}$ -complete (under many-to-one Karp reductions):

## DEFINITION 9.14

Define the quantifier  $\oplus$  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.<sup>2</sup> The language  $\oplus\text{SAT}$  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 known that a polynomial-time algorithm for  $\oplus\mathbf{P}$  implies that  $\mathbf{NP} = \mathbf{P}$ . However, such an algorithm does imply that  $\mathbf{NP} = \mathbf{RP}$  since  $\mathbf{NP}$  can be probabilistically reduced to  $\oplus\text{SAT}$ :

## THEOREM 9.15 (VALIANT-VAZIRANI THEOREM)

There exists a probabilistic polynomial-time algorithm  $A$  such that for every  $n$ -variable Boolean formula  $\varphi$

$$\varphi \in \text{SAT} \Rightarrow \Pr[A(\varphi) \in \oplus\text{SAT}] \geq \frac{1}{8n}$$

$$\varphi \notin \text{SAT} \Rightarrow \Pr[A(\varphi) \in \oplus\text{SAT}] = 0$$

To prove Theorem 9.15 we use the following lemma on pairwise independent hash functions:

## LEMMA 9.16 (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,

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

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 for every  $x \neq x'$ ,  $\Pr[h(x) = 0^k \wedge h(x') = 0^k] = p^2$ . Let  $N$  be the random variable denoting the number of  $x \in S$  satisfying  $h(x) = 0^k$ . Note that  $\mathbf{E}[N] = |S|p \in [\frac{1}{4}, \frac{1}{2}]$ . By the inclusion-exclusion principle

$$\Pr[N \geq 1] \geq \sum_{x \in S} \Pr[h(x) = 0^k] - \sum_{x < x' \in S} \Pr[h(x) = 0^k \wedge h(x') = 0^k] = |S|p - \binom{|S|}{2} p^2$$

and by the union bound we get that  $\Pr[N \geq 2] \leq \binom{|S|}{2} p^2$ . Thus

$$\Pr[N = 1] = \Pr[N \geq 1] - \Pr[N \geq 2] \geq |S|p - 2 \binom{|S|}{2} p^2 \geq |S|p - |S|^2 p^2 \geq \frac{1}{8}$$

where the last inequality is obtained using the fact that  $\frac{1}{4} \leq |S|p \leq \frac{1}{2}$ . ■

<sup>2</sup>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  $\bigoplus_{x \in \{0,1\}^n} \varphi(x) = \bigoplus_{x_1 \in \{0,1\}} \cdots \bigoplus_{x_n \in \{0,1\}} \varphi(x_1, \dots, x_n)$ .

**Proof of Theorem 9.15**

We now use Lemma 9.16 to prove Theorem 9.15. Given a formula  $\varphi$  on  $n$  variables, our probabilistic algorithm  $A$  chooses  $k$  at random from  $\{2, \dots, n+1\}$  and a random hash function  $h \in_R \mathcal{H}_{n,k}$ . It then uses the Cook-Levin reduction to compute a formula  $\tau$  on variables  $x \in \{0, 1\}^n, y \in \{0, 1\}^m$  (for  $m = \text{poly}(n)$ ) such that  $h(x) = 0$  if and only if there exists a *unique*  $y$  such that  $\tau(x, y) = 1$ .<sup>3</sup> The output of  $A$  if the formula

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

It is equivalent to the statement

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

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  $2^{k-2} \leq |S| \leq 2^k$ , conditioned on which, with probability  $1/8$ , 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 9.17 (HARDNESS OF UNIQUE SATISFIABILITY)

The proof of Theorem 9.15 implies the following stronger statement: the existence of an algorithm to 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  $\text{NP}$ . 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.

**9.3.2 Step 1: Randomized reduction from PH to  $\oplus\text{P}$** 

We now go beyond  $\text{NP}$  (that is to say, the Valiant-Vazirani theorem) and show that we can actually reduce any language in the polynomial hierarchy to  $\oplus\text{SAT}$ .

LEMMA 9.18

Let  $c \in \mathbb{N}$  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,

$$\begin{aligned} \psi \text{ is true} &\Rightarrow \Pr[A(\psi) \in \oplus\text{SAT}] \geq \frac{2}{3} \\ \psi \text{ is false} &\Rightarrow \Pr[A(\psi) \in \oplus\text{SAT}] = 0 \end{aligned}$$

Before proving the Lemma, let us make a few notations and observations: For a Boolean formula  $\varphi$  on  $n$  variables, let  $\#(\varphi)$  denote the number of satisfying assignments of  $\varphi$ . We consider also formulae  $\varphi$  that are *partially quantified*. That is, in addition to the  $n$  variables  $\varphi$  takes as input

<sup>3</sup>For some implementations of hash functions, such as the one described in Exercise 4, one can construct directly (without going through the Cook-Levin reduction) such a formula  $\tau$  that does not use the  $y$  variables.

it may also have other variables that are bound by a  $\forall, \exists$  or  $\bigoplus$  quantifiers (for example  $\varphi$  can be of the form  $\varphi(x_1, \dots, x_n) = \forall y \in \{0, 1\}^n \tau(x_1, \dots, x_n, y)$  where  $\tau$  is, say, a 3CNF Boolean formula).

Given two (possibly partially quantified) formulae  $\varphi, \psi$  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  $\#(\varphi \cdot \psi) = \#(\varphi)\#(\psi)$  and  $\#(\varphi + \psi) = \#(\varphi) + \#(\psi)$ . Indeed, take  $\varphi \cdot \psi(x, y) = \varphi(x) \wedge \psi(y)$  and  $\varphi + \psi(z) = ((z_0 = 0) \wedge \varphi(z_1, \dots, z_n)) \vee ((z_0 = 1) \wedge \psi(z_1, \dots, z_m))$ . For a formula  $\varphi$ , we 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 to a number flips the parity, for every two formulae  $\varphi, \psi$  as above

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

$$\neg \bigoplus_x \varphi(x) \Leftrightarrow \bigoplus_{x,z} (\varphi + 1)(x, z) \quad (4)$$

$$\left(\bigoplus_x \varphi(x)\right) \vee \left(\bigoplus_y \psi(y)\right) \Leftrightarrow \bigoplus_{x,y,z} ((\varphi + 1) \cdot (\psi + 1) + 1)(x, y, z) \quad (5)$$

**PROOF OF LEMMA 9.18:** Recall that membership in a  $\mathbf{PH}$ -language can be reduced to deciding the truth of a quantified Boolean formula with a constant number of alternating quantifiers. The idea behind the proof is to replace one-by-one each  $\exists/\forall$  quantifiers with a  $\bigoplus$  quantifier.

Let  $\psi$  be a formula with  $c$  levels of alternating  $\exists/\forall$  quantifiers, possibly with an initial  $\bigoplus$  quantifier. We 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, satisfying **(1)** if  $\psi$  is false then so is  $\psi'$ , and **(2)** if  $\psi$  is true then with probability at least  $1 - \frac{1}{10c}$ ,  $\psi'$  is true as well. The lemma follows by repeating this step  $c$  times.

For ease of notation, we demonstrate the proof for the case that  $\psi$  has a single  $\bigoplus$  quantifier and two additional  $\exists/\forall$  quantifiers. We can assume without loss of generality that  $\psi$  is of the form

$$\psi = \bigoplus_{z \in \{0,1\}^\ell} \exists_{x \in \{0,1\}^n} \forall_{w \in \{0,1\}^k} \varphi(z, x, w),$$

as otherwise we can use the identities  $\forall_x P(x) = \neg \exists_x \neg P(x)$  and (4) to transform  $\psi$  into this form.

The proof of Theorem 9.15 provides for every  $n$ , a probabilistic algorithm that outputs a formula  $\tau$  on variables  $x \in \{0, 1\}^n$  and  $y \in \{0, 1\}^m$  such that for every nonempty set  $S \subseteq \{0, 1\}^n$ ,  $\Pr[\bigoplus_{x \in \{0,1\}^n, y \in \{0,1\}^m} \tau(x, y)] \geq 1/(8n)$ . Run this algorithm  $t = 100c\ell \log n$  times to obtain the formulae  $\tau_1, \dots, \tau_t$ . Then, for every nonempty set  $S \subseteq \{0, 1\}^n$  the probability that there does not exist  $i \in [t]$  such that  $\bigoplus_{x \in \{0,1\}^n, y \in \{0,1\}^m} \tau_i(x, y)$  is TRUE is less than  $2^{-\ell}/(10c)$ . We claim that this implies that with probability at least  $1 - 1/(10c)$ , the following formula is equivalent to  $\psi$ :

$$\bigoplus_{z \in \{0,1\}^\ell} \theta(z), \quad (6)$$

where

$$\theta(z) = \vee_{i=1}^t \left( \bigoplus_{x \in \{0,1\}^n, y \in \{0,1\}^m} \forall_{w \in \{0,1\}^k} \tau_i(x, y) \wedge \varphi(x, z, w) \right)$$

Indeed, for every  $z \in \{0, 1\}^\ell$  define  $S_z = \left\{ x \in \{0, 1\}^n : \forall_{w \in \{0, 1\}^k} \varphi(x, z, w) \right\}$ . Then,  $\psi$  is equivalent to  $\bigoplus_{z \in \{0, 1\}^\ell} |S_z|$  is nonempty. But by the union bound, with probability at least  $1 - 1/(10c)$  it holds that for *every*  $z$  such that  $S_z$  is nonempty, there exists  $\tau_i$  satisfying  $\bigoplus_{x, y} \tau_i(x, y)$ . This means that for every such  $z$ ,  $\theta(z)$  is true. On the other hand, if  $S_z$  is empty then certainly  $\theta(z)$  is false, implying that indeed  $\psi$  is equivalent to (6).

By applying the identity (5), we can transform (6) into an equivalent formula of the desired form

$$\bigoplus_{z, x, y, w} \forall_w \varphi'(x, y, z, w)$$

for some unquantified polynomial-size formula  $\varphi'$ . ■

### 9.3.3 Step 2: Making the reduction deterministic

To complete the proof of Toda's Theorem (Theorem 9.11), we prove the following lemma:

LEMMA 9.19

*There is a (deterministic) polynomial-time transformation  $T$  that, for every formula  $\psi$  that is an input for  $\oplus\text{SAT}$ ,  $T(\psi, 1^m)$  is an unquantified Boolean formula and*

$$\begin{aligned} \psi \in \oplus\text{SAT} &\Rightarrow \#(\varphi) = -1 \pmod{2^{m+1}} \\ \psi \notin \oplus\text{SAT} &\Rightarrow \#(\varphi) = 0 \pmod{2^{m+1}} \end{aligned}$$

PROOF OF THEOREM 9.11 USING LEMMAS 9.18 AND 9.19.: Let  $L \in \mathbf{PH}$ . We show that we can decide whether an input  $x \in L$  by asking a single question to a  $\#\text{SAT}$  oracle. For every  $x \in \{0, 1\}^n$ , Lemmas 9.18 and 9.19 together imply there exists a polynomial-time TM  $M$  such that

$$\begin{aligned} x \in L &\Rightarrow \Pr_{r \in_R \{0, 1\}^m} [\#(M(x, r)) = -1 \pmod{2^{m+1}}] \geq \frac{2}{3} \\ x \notin L &\Rightarrow \forall_{r \in_R \{0, 1\}^m} \#(M(x, r)) = 0 \pmod{2^{m+1}} \end{aligned}$$

where  $m$  is the (polynomial in  $n$ ) number of random bits used by the procedure described in that Lemma. Furthermore, even in the case  $x \in L$ , we are guaranteed that for every  $r \in \{0, 1\}^m$ ,  $\#(M(x, r)) \in \{0, -1\} \pmod{2^{m+1}}$ .

Consider the function that maps two strings  $r, u$  into the evaluation of the formula  $M(x, r)$  on the assignment  $u$ . Since this function is computable in polynomial-time, the Cook-Levin transformation implies that we can obtain in polynomial-time a CNF formula  $\theta_x$  on variables  $r, u, y$  such that for every  $r, u$ ,  $M(x, r)$  is satisfied by  $u$  if and only if there exist a unique  $y$  such that  $\theta_x(r, u, y)$  is true. Let  $f_x(r)$  be the number of  $u, y$  such that  $\theta_x(r, u, y)$  is true, then

$$\#(\theta_x) = \sum_{r \in \{0, 1\}^m} f_x(r),$$

But if  $x \notin L$  then  $f_x(r) = 0 \pmod{2^{m+1}}$  for every  $r$ , and hence  $\#(\theta_x) = 0 \pmod{2^{m+1}}$ . On the other hand, if  $x \in L$  then  $f_x(r) = -1 \pmod{2^{m+1}}$  for between  $\frac{2}{3}2^m$  and  $2^m$  values of  $r$ , and is

equal to 0 on the other values, and hence  $\#(\theta_x) \neq 0 \pmod{2^{m+1}}$ . We see that deciding whether  $x \in L$  can be done by computing  $\#(\theta_x)$ . ■

**PROOF OF LEMMA 9.19:** For every pair of formulae  $\varphi, \tau$  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 formulae 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

$$\#(\tau) = -1 \pmod{2^{2^i}} \Rightarrow \#(4\tau^3 + 3\tau^4) = -1 \pmod{2^{2^{i+1}}} \quad (7)$$

$$\#(\tau) = 0 \pmod{2^{2^i}} \Rightarrow \#(4\tau^3 + 3\tau^4) = 0 \pmod{2^{2^{i+1}}} \quad (8)$$

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 (7), (8) 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  $\psi$ . ■

#### WHAT HAVE WE LEARNED?

- The class  $\#\mathbf{P}$  consists of functions that count the number of certificates for a given instance. If  $\mathbf{P} \neq \mathbf{NP}$  then it is not solvable in polynomial time.
- Counting analogs of many natural  $\mathbf{NP}$ -complete problems are  $\#\mathbf{P}$ -complete, but there are also  $\#\mathbf{P}$ -complete counting problems for which the corresponding decision problem is in  $\mathbf{P}$ . One example for this is the problem `perm` of computing the permanent.
- Surprisingly, counting is more powerful than alternating quantifiers: we can solve every problem in the polynomial hierarchy using an oracle to a  $\#\mathbf{P}$ -complete problem.
- The classes  $\mathbf{PP}$  and  $\oplus\mathbf{P}$  contain the decision problems that correspond to the most significant and least significant bits (respectively) of a  $\#\mathbf{P}$  function. The class  $\mathbf{PP}$  is as powerful as  $\#\mathbf{P}$  itself, in the sense that if  $\mathbf{PP} = \mathbf{P}$  then  $\#\mathbf{P} = \mathbf{FP}$ . We do not know if this holds for  $\oplus\mathbf{P}$  but do know that every language in  $\mathbf{PH}$  randomly reduces to  $\oplus\mathbf{P}$ .

## 9.4 Open Problems

- What is the exact power of  $\oplus\mathbf{SAT}$  and  $\#\mathbf{SAT}$  ?
- 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 ??).

DRAFT



## Chapter notes and history

The definition of  $\#\mathbf{P}$  as well as several interesting examples of  $\#\mathbf{P}$  problems appeared in Valiant's seminal paper [?]. The  $\#\mathbf{P}$ -completeness of the permanent is from his other paper [?]. Toda's Theorem is proved in [?]. The proof given here follows the proof of [?] (although we use formulas where they used circuits.)

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

## 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 ??).

**Hint:** without loss of generality you can think that  $f = \#\text{CKT} - \text{SAT}$ , 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  $\mathbf{FP}^{\#\text{SAT}}$ .
- §3 Complete the analysis of the XOR gadget in the proof of Theorem 9.8. Let  $G$  be any weighted graph containing a pair of edges  $\overrightarrow{uu'}$  and  $\overrightarrow{vv'}$ , and let  $G'$  be the graph obtained by replacing these edges with the XOR gadget. Prove that every cycle cover of  $G$  of weight  $w$  that uses exactly one of the edges  $\overrightarrow{uu'}$  is mapped to a set of cycle covers in  $G'$  whose total weight is  $4w$ , and all the other cycle covers of  $G'$  have total weight 0.
- §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  $\text{GF}(2)$ . For every  $k \times n$  matrix  $A$  with entries in  $\text{GF}(2)$ , and  $k$ -length vector  $b \in \text{GF}(2)^k$ ,  $\mathcal{H}_{n,k}$  contains the function  $h_{A,b} : \text{GF}(2)^n \rightarrow \text{GF}(2)^k$  defined as follows:  $h_{A,b}(x) = Ax + b$ .
- §5 Show that if there is a polynomial-time algorithm that approximates  $\#\text{CYCLE}$  within a factor  $1/2$ , then  $\mathbf{P} = \mathbf{NP}$ .
- §6 Show that if  $\mathbf{NP} = \mathbf{P}$  then for every  $f \in \#\mathbf{P}$  and there is a polynomial-time algorithm that approximates  $f$  within a factor of  $1/2$ . Can you show the same for a factor of  $1 - \epsilon$  for arbitrarily small constant  $\epsilon > 0$ ? Can you make these algorithms *deterministic*?

Note that we do not know whether  $\mathbf{P} = \mathbf{NP}$  implies that exact computation of functions in  $\#\mathbf{P}$  can be done in polynomial time.

**Hint:** Use hashing and ideas similar to those in the proof of Toda's theorem, where we also needed to estimate the size of a set of strings. If you find this question difficult you might want to come back to it after seeing the Goldwasser-Sipser set lowerbound protocol of Chapter 8. To make the algorithm deterministic use the ideas of the proof that  $\mathbf{BPP} \subseteq \mathbf{PH}$  (Theorem 7.18).

§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  $\text{poly}(\log n)$ .

**Hint:** use the proof of Lemma 9.18.