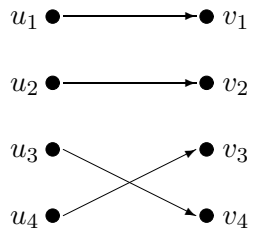
Chapter 7

Randomized Classes

7.1 Three examples of randomized algorithms

7.1.1 Perfect matching

Consider a bipartite graph G, and its adjacency matrix defined as follows: $(A_G)_{ij} = x_{ij}$ iff $(u_i, v_j) \in E_G$.



$$\begin{pmatrix} x_{11} & 0 & 0 & 0 \\ 0 & x_{22} & 0 & 0 \\ 0 & 0 & 0 & x_{34} \\ 0 & 0 & x_{43} & 0 \end{pmatrix}$$

Then, G has a *perfect matching* (i.e., each vertex on the left may be paired with a unique vertex on the right) iff $\det(A_G) = \Sigma_\sigma \text{sgn}(\sigma)\Pi_i(A_G)_{i\sigma(i)} \neq 0$.

Computing the symbolic determinant is computationally very expensive, so instead we randomly assign values to the x_{ij}'s. Let $A_G(x_1, \ldots, x_m)$, $m = |E_G|$, be A_G with its variables renamed to $x_1, \ldots, x_m$. Note that $m \leq n^2$ and each x_l represents some x_{ij}.

Choose m random integers $i_1, \ldots, i_m$ between 0 and $M = 2m$, and compute the integer determinant of $A_G(i_1, \ldots, i_m)$. Now,

if $\det(A_G(i_1, \ldots, i_m)) \neq 0$, then "yes", G has a perfect matching,
if $\det(A_G(i_1, \ldots, i_m)) = 0$, then "no", G *probably* has no perfect matching.

This is a polytime *Monte Carlo algorithm*, where "yes" answers are reliable and final, while "no" answers are in danger of a *false negative*. In this case

G might have a perfect matching, but unluckily $(i_1, \ldots, i_m)$ may happen to be roots of the polynomial $\det(A_G(x_1, \ldots, x_m))$.

Formally, N is a Monte Carlo TM for L if whenever $x \in L$, then at least half of the computations of N on x halt in "yes". If $x \notin L$, then all computations halt in "no". In other words, a Monte Carlo TM rejects "unanimously," and accepts "by majority."

Lemma 7.1.1 (Schwarz-Zippel) *Consider polynomials over $\mathbb{Z}$, and let $p(x_1, \ldots, x_m) \neq 0$ be a polynomial, where the degree of each variable is $\leq d$ (when the polynomial is written out as a sum of monomials), and let $M > 0$. Then the number of m-tuples $(i_1, \ldots, i_m) \in \{0, 1, \ldots, M-1\}^m$ such that $p(i_1, \ldots, i_m) = 0$ is $\leq mdM^{m-1}$.*

PROOF: Induction on m (the number of variables). If $m = 1$, $p(x_1)$ can have at most $d \leq dM$ many roots, by the Fundamental Theorem of Algebra.

Suppose the lemma holds for $(m-1)$, and suppose that $p(i_1, \ldots, i_m) = 0$. There are two cases to consider: express $p(x_1, \ldots, x_m)$ as $y_d x_m^d + \cdots + y_0 x_m^0$, where $y_i = y_i(x_1, \ldots, x_{m-1}) \in \mathbb{Z}[x_1, \ldots, x_{m-1}]$. The first case is the situation where $y_d = \cdots = y_0 = 0$, i.e., the "coefficients" of this polynomial are zero under some value assignment to the x_i's, and so p is zero under that assignment. The probability of this situation happening is certainly bounded above by the probability that $y_d = 0$ (i.e., only the first coefficient is zero under some value assignment). The second case is where $y_d \neq 0$, under some truth value assignment. Thus the probability that $p = 0$ is bounded above by the sum of the probabilities of these two cases.

We now compute the probability of each case. Case (1), the coefficient of the highest degree of x_m (this coefficient is in $\mathbb{Z}[x_1, \ldots, x_{m-1}]$) is zero. By the induction hypothesis, this coefficient is zero for at most $(m-1)dM^{m-2}$ many values, and x_m can take M values, and so the polynomial is zero for at most $(m-1)dM^{m-1}$ values. Case (2), for each combination of M^{m-1} values for $x_1, \ldots, x_{m-1}$, there are at most d roots of the resulting polynomial (again by the Fundamental Theorem of Algebra), i.e., dM^{m-1}. Adding the two estimates gives us mdM^{m-1}. □

We apply lemma 7.1.1 to the Monte Carlo algorithm for matching given above, with $M = 2m$, and obtain that the probability of a false negative is

$$\leq \frac{m \cdot d \cdot M^{m-1}}{M^m} = \frac{m \cdot 1 \cdot (2m)^{m-1}}{(2m)^m} = \frac{m}{2m} = \frac{1}{2}.$$

Now suppose we perform "many independent experiments," meaning that we perform the above algorithm k many times. Then, if the answer always

comes zero we know that the probability of error is $\leq \left(\frac{1}{2}\right)^k = \frac{1}{2^k}$. For $k = 100$, the error becomes *negligible*.

The integer determinant can be computed in NC^2 with Berkowitz's algorithm (this is shown in the Appendix, in section 8.5), so this means that perfect matching is in co-RNC^2 (see the definition of randomized circuit families on page 110). On the other hand, matching is in P; this can be easily seen by using a "max flow algorithm": add two new nodes s, t, and connect s to all the nodes in the left-column of the matching problem, and connect t to all the nodes in the right-column of the matching problem, and give each edge a capacity of 1, and ask if there is a flow $\geq n$ (where n is the number of nodes in each of the two components of the given bipartite graph) from s to t; see figure 7.1.

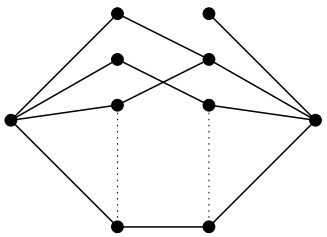


Figure 7.1: Reduction of matching to flow.

Counting the number of perfect matchings is complete for $\#\mathsf{P}$, while the decision problem itself is in P (it is usually the case, for problems which are complete for $\#\mathsf{P}$, such as #SAT, that the related decision problem is NP-complete).

The family of Boolean functions f_{pm} computing the existence of a perfect matching (where the input variables denote the presence or absence of edges) is monotone (adding new edges can only improve the chances of the existence of a perfect matching). On the other hand, as we noted before, perfect matching is in P, so we know that we can compute f_{pm} with polysize circuits. It was shown by Razborov, however, that monotone circuits computing f_{pm} are necessarily of exponential size (thus, removing the negation gate increases dramatically the circuit size necessary to compute f_{pm}). This also shows that giving an exponential lower bound for a monotone circuit family deciding a language in NP is not enough to show the separation of P and NP (see theorem 6.4.8).

One final observation is that perfect matching is not known to be complete for any natural complexity class.

7.1.2 Primality testing

A polytime algorithm for PRIMES $= \{(n)_b |\ n$ is primes $\}$ is now known (see [MA04]). But randomized algorithms for primality are much easier to describe (and to prove correct!), and more efficient.

Algorithm 7.1.2 (Rabin-Miller)
On input p:

1. If p is even, accept if $p = 2$; otherwise, reject.
2. Select $a_1, \ldots, a_k$ randomly in $\mathbb{Z}_p^+$.
3. For each i from 1 to k:
4. Compute $a_i^{(p-1)} \pmod p$ and reject if $\neq 1$.
5. Let $(p-1) = st$ where s is odd and $t = 2^h$.
6. Compute the sequence $a_i^{s\cdot 2^0}, a_i^{s\cdot 2^1}, a_i^{s\cdot 2^2}, \ldots, a_i^{s\cdot 2^h} \pmod p$.
7. If some element of this sequence is not 1,
   find the last element that is not 1,
   and reject if that element is not -1.
8. All tests passed, so accept at this point.

Note that if we got to line 6 in the algorithm, it means that $a_i^{s\cdot 2^h} = 1 \pmod p$. We say that a_i is a *witness* (of compositness) for p if the algorithm rejects at either stage 4 or stage 7, using a_i.

Lemma 7.1.3 *Suppose that p is a prime number. Then the Rabin-Miller algorithm accepts it with probability 1. (That is, there are no false negatives.)*

PROOF: We show that if p is prime, no witness exists, and so no branch of the algorithm rejects. If a were a stage 4 witness, $a^{(p-1)} \neq 1 \pmod p$ then Fermat's little theorem would imply that p is composite. If a were a stage 7 witness, some b exists in $\mathbb{Z}_p^+$, where $b \neq \pm 1 \pmod p$ and $b^2 = 1 \pmod p$. Therefore, $(b^2 - 1) = 0 \pmod p$. Factoring, we obtain that

$$(b-1)(b+1) = 0 \pmod p$$

which implies that p divides $(b-1)(b+1)$. But because $b \neq \pm 1 \pmod p$, both $(b-1)$ and $(b+1)$ are strictly between 0 and p. But that contradicts that $p|(b-1)(b+1)$, because p is a prime, so to divide the RHS it has to be a factor of the RHS, but both numbers are smaller than it. □

We are now going to show that the Rabin-Miller algorithm identifies composite numbers with high probability.

Lemma 7.1.4 *If p is an odd composite number then the probability that the Rabin-Miller algorithm accepts is $\leq 2^{-k}$. (Thus, for k sufficiently big, the probability of* false positives *is negligible.)*

PROOF: We show that if p is an odd composite number and a is selected randomly in $\mathbb{Z}_p^+$, then $\Pr[\ a \text{ is a witness }] \geq \frac{1}{2}$.

For every nonwitness, the sequence computed in stage 6 is either all 1s or contains a -1 at some position, followed by 1s. So, for example, 1 is a nonwitness of the first kind and -1 is a nonwitness of the second kind because s is odd and $(-1)^{s\cdot 2^0} = -1 \pmod p$ and $(-1)^{s\cdot 2^1} = 1 \pmod p$.

Among all nonwitnesses of the second kind, find a nonwitness for which the -1 appears in the largest position in the sequence. Let x be that nonwitness and let j be the position of -1 in its sequence, where the positions are numbered starting at 0. Hence $x^{s\cdot 2^j} = -1 \pmod p$ (and $x^{s\cdot 2^{j+1}} = 1 \pmod p$).

Because p is composite, either p is the power of a prime or we can write p as the product of q and r, two numbers that are co-prime. This yields two cases.

Case 1. Suppose that $p = q^e$ where q is prime and $e > 1$. Let $t = 1 + q^{e-1}$. From the binomial expansion of t^p we obtain:

$$t^p = (1+q^{e-1})^p = 1 + pq^{e-1} + \text{“multiples of higher power of } q^{e-1}\text{”} \quad (7.1)$$

which is congruent to 1 (mod p). Hence t is a stage 4 witness because, if $t^{p-1} = 1 \pmod p$, then $t^p = t \pmod p$, which from the observation about (7.1) is not possible. We use this one witness to get many others. If d is a (stage 4) nonwitness, we have $d^{p-1} = 1 \pmod p$, but then $dt \pmod p$ is a witness. Moreover, if d_1, d_2 are distinct nonwitnesses, then $d_1 t \neq d_2 t \pmod p$. Otherwise,

$$d_1 = d_1 \cdot t \cdot t^{p-1} = d_2 \cdot t \cdot t^{p-1} = d_2 \pmod p.$$

Thus the number of (stage 4) witnesses must be at least as large as the number of nonwitnesses.

Case 2. By the CRT there exists $t \in \mathbb{Z}_p$ such that

$$\begin{array}{ll} t = x & \pmod q \\ t = 1 & \pmod r \end{array} \quad \Rightarrow \quad \begin{array}{ll} t^{s\cdot 2^j} = -1 & \pmod q \\ t^{s\cdot 2^j} = 1 & \pmod r \end{array}$$

Hence t is a witness because $t^{s\cdot 2^j} \neq \pm 1 \pmod p$[1] but $t^{s\cdot 2^{j+1}} = 1 \pmod p$. Now that we have one witness, we can get many more, and show that dt

[1] To see why $t^{s\cdot 2^j} \neq \pm 1 \pmod p$ observe the following: suppose that $a = -1 \pmod q$ and $a = 1 \pmod r$, where $\gcd(q,r) = 1$. Suppose that $p = qr | (a+1)$, then $q | (a+1)$ and

(mod p) is a unique witness for each nonwitness d by making two observations.

First, $d^{s\cdot 2^j} = \pm 1 \pmod p$ and $d^{s\cdot 2^{j+1}} = 1 \pmod p$ owing to the way that j was chosen. Therefore $dt \pmod p$ is a witness because $(dt)^{s\cdot 2^j} \neq \pm 1 \pmod p$ and $(dt)^{s\cdot 2^{j+1}} = 1 \pmod p$.

Second, if d_1 and d_2 are distinct nonwitnesses, $d_1 t \neq d_2 t \pmod p$. The reason is that $t^{s\cdot 2^{j+1}} = 1 \pmod p$. Hence $t \cdot t^{s\cdot 2^{j+1}-1} = 1 \pmod p$. Therefore, if $d_1 t = d_2 t \pmod p$, then

$$d_1 = d_1 t \cdot t^{s\cdot 2^{j+1}-1} = d_2 t \cdot t^{s\cdot 2^{j+1}-1} = d_2 \pmod p$$

Thus in case 2., as well, the number of witnesses must be at least as large as the number of nonwitnesses. □

Thus we have a good randomized algorithm for primality: no false negatives, and false positives with probability $\leq \frac{1}{2^k}$. (This yields a Monte Carlo algorithm for composites: no false positives, and false negatives with probability $\leq \frac{1}{2^k}$.)

7.1.3 Pattern matching

In this section we design a randomized algorithm for pattern matching. Consider the set of strings over $\{0,1\}$, and let $M : \{0,1\} \longrightarrow M_{2\times 2}(\mathbb{Z})$, that is, M is a map from strings to 2×2 matrices over the integers ($\mathbb{Z}$) defined as follows:

$$M(\varepsilon) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$M(0) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$
$$M(1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

and for strings $x, y \in \{0,1\}^*$, $M(xy) = M(x)M(y)$, where the operation on the LHS is concatenation of strings, and the operation on the RHS is multiplication of matrices.

First of all, $M(x)$ is well defined because matrix multiplication is associative, and second of all, $M(x) = M(y)$ implies that $x = y$ (i.e., the map M is 1-1). Given $M = M(x)$ we can "decode" x uniquely as follows: if the

$r|(a+1)$, and since $r|(a-1)$ as well, it follows that $r|[(a+1)-(a-1)]$, so $r|2$, so $r = 2$, so p must be even, which is not possible since we deal with even p's in line 1 of the algorithm.

first column of M is greater than the second (where the comparison is made component-wise), then the last bit of x is zero, and otherwise it is 1. Let M' be M where we subtract the smaller column from the larger, and repeat.

For $x \in \{0,1\}^n$, the entries of $M(x)$ are bounded by Fibonacci number F_n. Let $F_0 = F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n > 1$. For a given string x, $M(x_1 x_2 \dots x_n)$ is such that the "smaller" column is bounded by F_{n-1} and the "larger" column is bounded by F_n. We can show this inductively: the basis case, $x = x_1$, is obvious. For the inductive step, assume it holds for $x \in \{0,1\}^n$, and show it still holds for $x \in \{0,1\}^{n+1}$: this is obvious as whether x_{n+1} is 0 or 1, one column is added to the other, and the other column remains unchanged.

By considering the matrices $M(x)$ modulo a suitable prime p, we perform efficient randomized pattern matching. We wish to determine whether x is a substring of y, where $|x| = n$, $|y| = m$, $n \leq m$. Let $y(i) = y_i y_{i+1} \dots y_{n+i-1}$, for appropriate i's. Select a prime $p \in \{1, \dots, nm^2\}$, and let $A = M(x) \pmod p$ and $A(i) = M(y(i)) \pmod p$. Note that

$$A(i+1) = M^{-1}(y_i) A(i) M(y_{n+i}) \pmod p,$$

which makes the computation of subsequent $A(i)$'s efficient.

So for all appropriate i's, we check whether $A = A(i)$. If yes, we check whether we did not get a false positive using a bit-by-bit comparison. If they match, we answer "yes", otherwise we change the prime p and continue[2].

What is the probability of getting a false positive? It is the probability that $A(i) = M(y(i)) \pmod p$ even though $A(i) \neq M(y(i))$. This is less than the probability that $p \in \{1, \dots, nm^2\}$ divides a (non-zero) entry in $A(i) - M(y(i))$. Since these entries are bounded by $F_n < 2^n$, less than n distinct primes can divide any of them. On the other hand, there are $\pi(nm^2) \approx (nm^2)/(\log(nm^2))$ primes in $\{1, \dots, nm^2\}$. So the probability of a false positive is $O(1/m)$.

Note that this algorithm has no error; it is randomized, but all potential answers are checked for false positives. Checking for these potential candidates is called *fingerprinting*. The randomness lowers the average time complexity of the procedure.

[2]For more details on this question, and in particular for an interesting discussion of why we need to change the prime after a false positive, see [KR87].

7.2 Basic Randomized Classes

In this section we present four standard randomized classes: RP, ZPP, BPP, and PP. We assume throughout that N is a nondeterministic polytime TM, where at each step there are exactly two nondeterministic choices (i.e., the degree of nondeterminism is always 2), and N is *precise*, i.e., all computations on x halt after the same number of steps, $p(|x|)$.

Let RP be the class of languages with polytime Monte Carlo TMs (see page 102). Note that $\mathsf{P} \subseteq \mathsf{RP} \subseteq \mathsf{NP}$.

RP is a *semantic class*, as opposed to a *syntactic class*. The intuition is that by "examining" a TM, in general we cannot tell if it is a Monte Carlo machine. On the other hand, syntactic classes are such that we can enumerate all the TMs which decide the class; examples of syntactic classes are P and NP, since we can list all polytime deterministic (or nondeterministic) TMs (see page 45 where we list all (oracle) polytime deterministic TMs). The problem with semantic classes is that usually no known complete problems exist for them. On the other hand, every syntactic class $\mathcal{M}$ (e.g., nondeterministic polytime) has the following hard problem

$$\{\langle M, x\rangle |\ M \in \mathcal{M} \text{ and } M(x) = \text{"yes"}\ \}.$$

which for P and NP also turns out to be complete. For semantic classes, this language is usually undecidable.

Is RP closed under complement? The class co-RP has false positives, but not false negatives. In section 7.1.2 we showed that PRIMES $\in$ co-RP, although it is now known that PRIMES $\in$ P.

Consider $\mathsf{RP} \cap \text{co-RP}$. Any problem in this class has two Monte Carlo algorithms, one with no false positives, and the other with no false negatives. If we execute both algorithms independently k times, the probability of not obtaining a definitive answer falls down to 2^{-k}, because it means that one of them is giving the wrong answer k times in the row (see figure 7.2). Let $\mathsf{ZPP} = \mathsf{RP} \cap \text{co-RP}$. The class ZPP is a randomized class with zero probability of error and an expected polynomial running time. A ZPP algorithm is called a *Las Vegas algorithm*. At the time of writing these notes, no algorithm class has been named after Baden-Baden.

The most comprehensive but still plausible notion of realistic computation is the class BPP (Bounded Probability of error and Polynomial running time). This class consists of languages L for which there is a TM N with the following properties: if $x \in L$, then at least $\frac{3}{4}$ of computations accept, and if $x \notin L$, then at least $\frac{3}{4}$ of computations reject. (In other words, acceptance and rejection by *clear majority*.)

	$x \in L$	$x \notin L$
RP	$\geq 1/2$ "yes"	all "no"
co-RP	all "no"	$\geq 1/2$ "yes"

Figure 7.2: ZPP.

We say that a language is in PP if there is a nondeterministic polytime TM N (precise—as always when considering randomized algorithms) such that $\forall x$, $x \in L$ iff more than half of the computations of N on input x end up accepting. We say that N decides L "by majority" (as opposed to "by clear majority" as in the case of BPP). PP is a syntactic class since every nondeterministic precise TM N can be used to define a language in PP.

Note that we have:

$$\mathsf{RP} \subseteq \mathsf{BPP} = \text{co-}\mathsf{BPP} \subseteq \mathsf{PP},$$

where the first inclusion follows from the fact that we can run an RP algorithm twice to get the probability of false negatives down to $\leq \frac{1}{4}$, and the second inclusion follows from the fact that a "clear majority" implies a "slim majority."

A fundamental open problem is whether $\mathsf{BPP} \stackrel{?}{=} \mathsf{P}$ or even $\mathsf{BPP} \stackrel{?}{\subseteq} \mathsf{NP}$.

Exercise 7.2.1 *Show that the classes* RP, BPP*, and* PP *are closed under* $\leq^m_{\mathsf{L}}$ *(logspace many-one reductions).*

Exercise 7.2.2 *Show that* BPP *and* RP *are closed under union and intersection. Also show that* PP *is closed under complement and symmetric difference.*

Exercise 7.2.3 *Show that if* $\mathsf{NP} \subseteq \mathsf{BPP}$ *then* $\mathsf{RP} = \mathsf{NP}$*.*

Exercise 7.2.4 *Show that* MajSat *(consisting of* ϕ*'s for which more than half of the truth assignments to* $Var(\phi)$ *are satisfying) is* PP*-complete.*

Lemma 7.2.5 $\mathsf{NP} \subseteq \mathsf{PP}$*.*

Proof: Suppose that L is in NP, decided by a precise N. Then the following N' decides L by majority: add a new initial state to N, and a nondeterministic choice out of it, one to the old initial state which leads to the old computation (i.e., the computation of the original N), and the other leads

to a new precise computation of the same length as the old one, where all leaves are accepting[3]. □

A family of circuits $\{C_n\}$ is a *randomized circuit family for f* if in addition to the n inputs $x_1, x_2, \ldots, x_n$, it takes m (random) bits $r_1, r_2, \ldots, r_m$, and furthermore C_n satisfies two properties:

- If $f_n(x_1, \ldots, x_n) = 0$, then $C_n(x_1, \ldots, x_n, r_1, \ldots, r_m) = 0$ *regardless* of the value of the r_i's (i.e., no false positives).
- If $f_n(x_1, \ldots, x_n) = 1$, then $C_n(x_1, \ldots, x_n, r_1, \ldots, r_m) = 1$ with probability at least $1/2$.

This is the nonuniform version of a Monte Carlo algorithm, i.e., the nonuniform version of RP. Let RNC^i to be the class of languages recognizable by randomized NC^i circuit families (see page 103 where we claimed that perfect matching is in RNC^2).

In fact we can derandomize Monte Carlo circuit families (at the cost of losing uniformity—if our circuit family was uniform).

Theorem 7.2.6 *If f has a randomized polysize circuit family, then it has a polysize circuit family (i.e., it can be* derandomized*).*

PROOF: For each n we form a matrix M with 2^n rows, corresponding to each input, and 2^m columns, corresponding to each random input. Let M_{jk} be 1 if the random input corresponding to the k-th column is a *witness* to the input corresponding to the j-th input (i.e., this choice of random bits sets the circuit on that input to be 1). Eliminate all rows for which f_n is zero. At least half of the entries in each surviving row are 1, so there must be a column where at least half of the entries are 1.

Pick the witness $\vec{\mathbf{r}}_1 = r_1, \ldots, r_m$ corresponding to this column; it is a witness for at least half of the inputs, so delete the rows corresponding to those inputs, and repeat.

At each stage, we find a witness to at least half of the remaining inputs, so after $\log(2^n) = n$ many steps we will find witnesses $\vec{\mathbf{r}}_1, \ldots, \vec{\mathbf{r}}_n$ for all the inputs. Now let $C'_n(x_1, \ldots, x_n)$ be the `Or` of the circuits $C_n(x_1, \ldots, x_n, \vec{\mathbf{r}}_i)$, $i = 1, \ldots, n$. □

[3]When showing relationships between the different randomized classes one often has to modify the number of accepting and rejecting paths. When doing this, it is important to remember that the different nondeterministic branches are "not aware of each other." So the changes can not be made dependent of the contents of the individual branches.

7.3 The Chernoff Bound and Amplification

Suppose that L is in RP, and A is the Monte Carlo algorithm that decides L. We run A on x (independently) $p(|x|)$-many times. If the answer always come "no", we answer "no". The first time a "yes" answer comes, we answer "yes" and stop running A. If $x \notin L$, the answer will always be "no", and we answer correctly. If $x \in L$, then the probability that we answer "no" at the end is $\leq \left(\frac{1}{2}\right)^{p(|x|)}$, i.e., exponentially small! This is called *amplification* or *error reduction*; at a polynomial cost, it increases the probability of getting the right answer exponentially: *"From fairest creatures we desire increase"* (Shakespeare, Sonnet I).

Lemma 7.3.1 (Chernoff Bound) *Suppose that $X_1, X_2, \ldots, X_n$ are independent random variables, taking the value 1 and 0 with probability p and $(1-p)$, respectively, and consider $X = \sum_{i=1}^{n} X_i$. Then, for $0 < \theta \leq 1$,*

$$\Pr[X \geq (1+\theta)pn] \leq e^{-\frac{\theta^2}{3}pn}.$$

Proof:

$$\Pr[X \geq (1+\theta)pn] = \Pr[e^{tX} \geq e^{t(1+\theta)pn}] \leq e^{-t(1+\theta)pn} E(e^{tX})$$

where $t \in \mathbb{R}^+$, and where we applied a version of Markov's inequality (which states that $\Pr[X \geq kE(X)] \leq \frac{1}{k}$; see the proof of claim 5.3.10, on page 69), with $k = e^{t(1+\theta)pn} E(e^{tX})^{-1}$.

Since $X = \sum_{i=1}^{n} X_i$, where the X_i are independent random variables, we have that

$$E(e^{tX}) = (E(e^{tX_1}))^n = (1 \cdot (1-p) + e^t \cdot p)^n = (1 + p(e^t - 1))^n.$$

Therefore,

$$\Pr[X \geq (1+\theta)pn] \leq e^{-t(1+\theta)pn}(1 + p(e^t - 1))^n.$$

Now note that $(1+y) \leq e^y$ (see the figure 7.3), and so $(1 + p(e^t - 1)) \leq e^{p(e^t-1)}$, and so we can conclude that

$$\Pr[X \geq (1+\theta)pn] \leq e^{-t(1+\theta)pn} e^{pn(e^t-1)},$$

and $t = \ln(1+\theta)$ minimizes the RHS of the above equation, which finally gives us:

$$\Pr[X \geq (1+\theta)pn] \leq e^{pn(\theta - (1+\theta)\ln(1+\theta))}$$

and $(\theta - (1+\theta)\ln(1+\theta)) \leq \frac{-\theta^2}{3}$. □

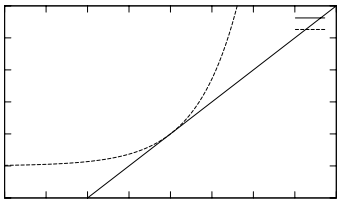


Figure 7.3: Functions e^y (dotted line) and $(1+y)$.

Corollary 7.3.2 *If $p = \frac{1}{2}+\varepsilon$ for some $0 < \varepsilon < \frac{1}{4}$, then $\Pr[X \leq \frac{n}{2}] \leq e^{-\frac{\varepsilon^2 n}{6}}$.*

PROOF: $X = \sum_{i=1}^{n} X_i \leq \frac{n}{2}$ iff $-\sum_{i=1}^{n} X_i \geq -\frac{n}{2}$ iff $n - \sum_{i=1}^{n} X_i \geq n - \frac{n}{2}$ iff $\sum_{i=1}^{n} \underbrace{(1 - X_i)}_{X'_i} \geq \frac{n}{2}$. Thus, X'_i is 1 with probability $p' = \frac{1}{2} - \varepsilon$, and 0 with probability $(1-p')$, and so now we apply the Chernoff bound (lemma 7.3.1) and obtain

$$\Pr[X' \geq (1+\theta)(\frac{1}{2} - \varepsilon)n] \leq e^{-\frac{\theta^2}{3}(\frac{1}{2}-\varepsilon)n}$$

To make $(1+\theta)(\frac{1}{2} - \varepsilon) = \frac{1}{2}$, we have to set $\theta = \frac{\varepsilon}{\frac{1}{2}-\varepsilon}$, and now

$$\Pr[X \leq \frac{n}{2}] = \Pr[X' \geq \frac{n}{2}] \leq e^{-\frac{\left(\frac{\varepsilon}{\frac{1}{2}-\varepsilon}\right)^2}{3}(\frac{1}{2}-\varepsilon)n} < e^{-\frac{\varepsilon^2 n}{6}}$$

where the right-most inequality follows from basic algebra. □

Here are some consequences of this corollary.

- We can detect a bias ε in a coin with reasonable certainty by performing about $n = O(\frac{1}{\varepsilon^2})$ many experiments. Say, we let $n = 60\frac{1}{\varepsilon^2}$. Then $e^{-\frac{\varepsilon^2 n}{6}} < 0.00004$, so the probability that the side with the bias $\frac{1}{2} + \varepsilon$ does not show up in the majority of trials is very small.

- PP is not a "good" randomized class because there the bias ε may be equal to $2^{-p(n)}$, where p is a polynomial (i.e., the majority may be very slim, so if $x \in L$, only half plus 1 are "yes" computations, and half minus 1 are "no" computations, and if $x \notin L$ vice-versa). If ε is so slim, the algorithm then has to be run exponentially many times to get the correct answer with any reasonable degree of confidence.

- We could define BPP to have $\frac{1}{2}+\varepsilon$ as the probability of getting the right answer, even for very small ε. Suppose $0 < \varepsilon < \frac{1}{4}$. Suppose that N decides L by majority $\frac{1}{2}+\varepsilon$. Run N n-many times and accept as outcome the majority of outcomes. By choosing n suitably large we can bound the error by $\frac{1}{4}$: by corollary 7.3.2 we have

$$\Pr[\text{error}] = \Pr[X \leq \frac{n}{2}] \leq e^{-\frac{\varepsilon^2 n}{6}}.$$

  We want $e^{-\frac{\varepsilon^2 n}{6}} < \frac{1}{4}$, so it is enough to have $\frac{\varepsilon^2 n}{6} > 2$, i.e., $\varepsilon^2 n > 12$, i.e., $n = \lceil \frac{12}{\varepsilon^2} \rceil$. Note that ε does not even have to be a constant—it could be any inverse polynomial. We can do even better; say we want $e^{-\frac{\varepsilon^2 n}{6}} < \frac{1}{4^m}$, then just let n be greater than $(8/\varepsilon^2)m$.

  In other words, we can always assume that if we have a BPP algorithm for deciding a language, we have a BPP algorithm for deciding the same language where the probability of error is bounded by $\frac{1}{2^{|x|}}$ for any input x. This negligible probability of error (2^{100} is the estimated number of atoms in the observable universe) is what motivates the conjecture that P = BPP.

  Also, in the absence of a proof that P = BPP, this small probability of error suggest that perhaps BPP can replace P as the class of languages that can be recognized feasibly.

7.4 More on BPP

The following theorems show that we can replace randomness with nonuniformity.

Theorem 7.4.1 *All the languages in* BPP *have polysize circuits, that is* BPP $\subseteq$ P/poly.

PROOF: $L \in$ BPP, so it is decided by a nondeterministic N by clear majority, say $\frac{1}{4}$ probability of error. We show how to build $C = \langle C_n \rangle$ using the "probabilistic method in combinatorics"[4]. Let $p(n)$ be the length of computation of N on inputs of length n (as always, assume N is precise). Consider a sequence $A_n = \{a_1, \ldots, a_m\}$ of bit strings, each $a_i \in \{0,1\}^{p(n)}$,

[4]Our first example of this method was the lower bound for parity, section 5.3.1 (page 65).

and let $m = 12(n+1)$. Each a_i represents a particular branch of N on an input of length n.

Define $C_n(x)$ to be the majority of outcomes of $N_{a_1}(x), \ldots, N_{a_m}(x)$, where N_{a_i} is a polytime deterministic Turing machine obtained from N by taking the branch specified by a_i, so $N_{a_i}(x)$ can be simulated with a polysize circuit. We argue that for every n there exists an A_n so that C_n works correctly on all inputs x.

Claim 7.4.2 *$\forall n > 0$, $\exists A_n$, $|A_n| = m = 12(n+1)$, such that for any given x of length n, less than half of the strings in A_n are* bad *for x (a_i is "bad for x" if $N_{a_i}(x) \neq L(x)$).*

PROOF: Generate A_n randomly. We show that the probability that for each $x \in \{0,1\}^n$ more than half the strings in A_n are good is at least $\frac{1}{2}$. Thus, an A_n with at least half the strings being good exists.

For each $x \in \{0,1\}^n$, at most $\frac{1}{4}$ of the computations are bad. So the expected number of bad a_i's is $\frac{m}{4}$. We use the Chernoff bound (lemma 7.3.1) with the following parameters:

$$X = \sum_{i=1}^{m} X_i \text{ where } X_i = \begin{cases} 1 & a_i \text{ is bad} \\ 0 & a_i \text{ is good} \end{cases}$$
$$\theta = 1$$
$$p = \frac{1}{4} = \text{probability of } a_i \text{ being bad}$$
$$m = |A_n| = 12(n+1),$$

to obtain

$$\Pr[X \geq (1+\theta)pm] \leq e^{\left(-\frac{\theta^2}{3}pm\right)},$$

and hence $\Pr[\underbrace{X \geq \frac{m}{2}}_{\left[\begin{smallmatrix}\text{prob. } m/2 \text{ or} \\ \text{more } a_i\text{'s are bad}\end{smallmatrix}\right]}] \leq e^{\left(-\frac{m}{12}\right)} < \frac{1}{2^{n+1}}$. Thus, the probability that for *some* $x \in \{0,1\}^n$ at least half of A_n are bad a_i's is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$. □ □

Note that we do not know an efficient way of constructing this A_n, for every n, since otherwise we would have shown that $\mathsf{BPP} = \mathsf{P}$.

Theorem 7.4.3 (Sipser) $\mathsf{BPP} \subseteq \Sigma_2^p$.

PROOF: Let $L \in \mathsf{BPP}$. By amplification we know that we can make the probability of error less than $\frac{1}{2^n}$, so let N be the amplified BPP machine

deciding L. Let $A(x) \subseteq \{0,1\}^{p(n)}$ be the set of accepting computations for a given x (i.e., strings representing a sequence of choices that lead to an accepting configuration—remember that our machines are always precise and of degree of nondeterminism exactly 2). We know the following:

- If $x \in L$, then $|A(x)| \geq 2^{p(n)}(1 - \frac{1}{2^n})$, and
- if $x \notin L$, then $|A(x)| \leq 2^{p(n)} \frac{1}{2^n}$.

Let $U = \{0,1\}^{p(n)}$, i.e., U is the set of bit strings of length $p(n)$. For $a, b \in U$, define $a \oplus b$ to be the bit-wise XOR of a, b, i.e., the sting $(a_1 \oplus b_1) \cdots (a_{p(n)} \oplus b_{p(n)})$. Note that $a \oplus b = c \iff c \oplus b = a$. Thus, for a given fixed b, $f_b(a) = a \oplus b$ is a one-to-one function, and f_b is a convolution, i.e., $f_b^2 = \mathrm{id}$. Further, if $r \in \{0,1\}^{p(n)}$ is a random string, then $f_b(r) = r \oplus a$ is also a random string (because f_b is just a permutation of U).

For any $t \in U$, let $A(x) \oplus t := \{a \oplus t | a \in A(x)\}$. Call this the *translation of $A(x)$ by t*, and note that $|A(x) \oplus t| = |A(x)|$.

Claim 7.4.4 *If $x \in L$, we can find a small set of translations of $A(x)$ that covers all of U.*

PROOF: Suppose $x \in L$, and consider $t_1, \ldots, t_{p(n)} \in U$, obtained uniformly at random. Fix $b \in U$. These translations *cover* b if $\exists j \leq p(n)$ such that $b \in A(x) \oplus t_j$.

$$\Pr[b \notin A(x) \oplus t_j] = \Pr[b \oplus t_j \notin A(x)] \leq \frac{1}{2^n}$$

since $b \in A(x) \oplus t_j \iff b \oplus t_j \in A(x)$, and since $x \in L$. So the probability that b is not covered by any of the t_j's is less than $2^{-n \cdot p(n)}$, and so the probability that some b in U is not covered is at most $2^{p(n)} 2^{-n \cdot p(n)}$, and so with overwhelming probability all of U is covered, and so there is a particular $T = \{t_1, \ldots, t_{p(n)}\}$ that covers U. □

On the other hand, if $x \notin L$, then $A(x)$ is an exponential fraction of U, so no polysize set T that covers U can exist. Therefore, there is a sequence T of $p(n)$ translations that cover U iff $x \in L$. Thus, L is the set of strings x such that: $\exists T \in \{0,1\}^{p(n)^2}$, such that $\forall b \in U$, there is a $j \leq p(n)$ such that $b \oplus t_j \in A(x)$. Since the last "there is" can be expressed as an `Or` of polynomially many things, and so can be made part of a polytime relation, we obtain a Σ_2^p predicate. □

Corollary 7.4.5 $\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$.

PROOF: Since BPP = co-BPP, the corollary follows directly from theorem 7.4.3. □

Corollary 7.4.6 *If* P = NP *then* P = BPP.

PROOF: If P = NP, then by theorem 4.3.2 the polytime hierarchy PH collapses to P, so by theorem 7.4.3, $\mathsf{BPP} \subseteq \Sigma_2^p \subseteq \mathsf{P}$. Since trivially $\mathsf{P} \subseteq \mathsf{BPP}$, the corollary follows. □

7.5 Toda's Theorem

Toda's theorem states that $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$, which can actually be stated more tightly as follows: $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$. (#P[1] means that we access the #P oracle only once; see page 19 for a definition of #P.) What this theorem says is that counting is more powerful than any constant number of alternations of quantifiers.

We introduce one more complexity class: $\oplus\mathsf{P}$, Parity P, the class of languages which are decidable by the parity of the number of accepting computations of a nondeterministic polytime TM. The class $\oplus\mathsf{P}$ contains, for example, the graph isomorphism problem.

The proof of Toda's result has two parts.

Part 1. $\forall k$, $\Sigma_k^p \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$; the consequence of this is that $\mathsf{PH} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$.

Part 2. $\mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$; the consequence of this is that $\mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$.

Putting the two parts together, we obtain that $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$ (since $\mathsf{BPP} \subseteq \mathsf{PP}$). On the other hand, $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$ (lemma 7.5.7), so the advertised result follows. We make some observations:

- Since the permanent (over 0-1 matrices computed over $\mathbb{Z}$) is #P-complete (as is #SAT), we could have restated Toda's theorem as claiming that PH fits in P with a single access to an oracle for computing the permanent.

- The permanent can be used to compute the number of perfect matchings in a bipartite graph, and it was already noted that the number of perfect matchings is #P complete (see page 103), while the related decision problem (i.e., *is there* a perfect matching in a given graph?) is in P. This is not the case for the decision problem related to #SAT, i.e., SAT, which is NP-complete.

To show Part 1, we use induction on k. The Basis Case is that $\Sigma_1^p = \mathsf{NP} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$ (which is lemma 7.5.1). Here is the plan for the proof of the

induction hypothesis:

$$\Sigma^p_{k+1} \overset{(1)}{=} \mathsf{NP}^{\Sigma^p_k} \overset{(2)}{\subseteq} \mathsf{NP}^{\mathsf{BPP}^{\oplus\mathsf{P}}} \overset{(3)}{\subseteq} \mathsf{BPP}^{\oplus\mathsf{P}^{\mathsf{BPP}^{\oplus\mathsf{P}}}} \overset{(4)}{\subseteq} \mathsf{BPP}^{\mathsf{BPP}^{\oplus\mathsf{P}^{\oplus\mathsf{P}}}} \overset{(5)}{\subseteq} \mathsf{BPP}^{\oplus\mathsf{P}}, \tag{7.2}$$

where (1) is by definition, (2) is by the induction hypothesis, (3) is the relativized version of the basis case, i.e., by lemma 7.5.1 ("Relativized Inclusion", $\mathsf{NP}^O \subseteq \mathsf{BPP}^{\oplus\mathsf{P}^O}$), (4) is by lemma 7.5.5 (i.e., "Swapping", $\oplus\mathsf{P}^{\mathsf{BPP}^O} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}^O}$), and (5) is by lemma 7.5.3 (i.e., "Collapsing", $\mathsf{BPP}^{\mathsf{BPP}^O} = \mathsf{BPP}^O$ and $\oplus\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}$).

Part 2 follows from the single lemma 7.5.6.

Lemma 7.5.1 *For any oracle O, $\mathsf{NP}^O \subseteq \mathsf{BPP}^{\oplus\mathsf{P}^O}$.*

PROOF: In fact we prove something stronger: $\mathsf{NP} \subseteq \mathsf{RP}^{\oplus\mathsf{P}}$ (or even stronger than that: $\mathsf{NP} \subseteq \mathsf{RP}^{\mathsf{UP}}$)[5]. Suppose that $L \in \mathsf{NP}$, so there exists a polytime language A such that $x \in L \iff \mathcal{F}(x) := \{y | \langle x, y\rangle \in A\}$ is non-empty. Consider the language

$$B = \{\langle x, W, j\rangle : |\{y \in \mathcal{F}(x) : W(y) = j\}| = 1\}. \tag{7.3}$$

Note that $W(y) = \sum_{i,y(i)=1} W(i)$ is a weight function, which can be encoded efficiently as a string which gives values (in an appropriate polysize range) to all the elements in the set $\{1, 2, \ldots, |y|\}$. In what follows, we are going to apply the Isolation Lemma (see the Appendix, section 8.4).

There exists a polynomial $q(n)$ such that

- If $x \in L$, $\Pr_{W,j}[\langle x, W, j\rangle \in B] \geq \frac{1}{q(|x|)}$, where W, j are bounded by an appropriate polynomial. To see this, note that with probability $\geq \frac{3}{4}$ a W is selected such that $|\{y \in \mathcal{F}(x) | W(y) = j\}| = 1$, *for some* j. Now a j is selected independently at random, among polynomially many values, so we have $\frac{1}{q(|x|)}$ instead.
- If $x \notin L$, then $\Pr = 0$.

Exercise 7.5.2 *Give precise bounds in the above outline, so that the analysis of the probabilities works out (in particular, what is the size of the range of W?).*

[5] UP, **U**nambiguous **P**olytime, is the class of languages decidable by a nondeterministic polytime TM where there are either no accepting paths or a single accepting path. Note that $\mathsf{P} \neq \mathsf{UP}$ iff worst-case one way functions exist.

If we replace the "= 1" in (7.3) with "is an odd number", then $B \in \oplus\mathsf{P}$. Now on input x, do $q(|x|)$-many independent trials of selecting W and j, and for each such pair query $\langle x, W, j\rangle \in B$, and accept iff at least one such trial is successful (in the sense that $\langle x, W, j\rangle \stackrel{?}{\in} B$ returns a "yes"). If $x \notin L$, clearly $\Pr[\text{accept}] = 0$, and if $x \in L$, then

$$\Pr[\text{reject}] < \left(1 - \frac{1}{q(|x|)}\right)^{q(|x|)} \uparrow_{x\to\infty} \frac{1}{e} < \frac{1}{2}.$$

The same argument can be repeated with an oracle O to obtain the result as advertised in the statement of the lemma. □

Lemma 7.5.3 *For any oracle O, $\mathsf{BPP}^{\mathsf{BPP}^O} = \mathsf{BPP}^O$, and $\oplus\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}$.*

PROOF: $\mathsf{BPP}^{\mathsf{BPP}} = \mathsf{BPP}$ follows by amplification: if we make the error exponentially small, and then directly simulate the oracle queries in the computation, then the number of incorrect paths (i.e., the paths on which some of the polynomially many oracle queries gave the wrong answer) is a small fraction of all the paths.

Exercise 7.5.4 *Formalize the above paragraph with appropriate parameters and show precise bounds.*

To show $\oplus\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}$, observe first that $\oplus\mathsf{P}$ is closed under complementation (just add one more choice from the initial state that lands immediately in an accepting state; this way the total number of accepting states increases by 1 and the parity flips)[6].

Now suppose that $L \in \oplus\mathsf{P}^{\oplus\mathsf{P}}$, so there is an oracle TM M^B such that $x \in L \iff$ the number of accepting paths of M^B on x is odd, where $B \in \oplus\mathsf{P}$. Let N_0 witness B and N_1 witness $\overline{B}$. Let M' be a nondeterministic TM which on input x, simulates M, except when M is about to make a query on y, M' guesses the answer $b \in \{0, 1\}$, and then simulates N_b on y. At the end, M' accepts on a path, if all the computations of N_0, N_1, and M are accepting on that path.

Now, for each accepting branch that ignores the computations of N_0, N_1 (i.e., we only take into account the guess b and continue immediately simulating M; call those branches "trunks"), there was a sequence of queries $y_1, y_2, \ldots, y_n$ on that path, and hence it gets blown up by $\Pi_{i=1}^n \#\text{accept}_{N_i}(y_i)$. Note that this number is odd iff all the n guesses are correct. All the trunks that correspond to bad guesses contribute an even factor. □

[6]Another way to see it is using the fact that #P is closed under addition: if N_0 is a polytime nondeterministic machine, then let N_1 be such that $\#\text{accept}_{N_1} = \#\text{accept}_{N_0} + 1$.

Lemma 7.5.5 *For any oracle O, $\oplus\mathsf{P}^{\mathsf{BPP}^O} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}^O}$.*

PROOF: We show that $\oplus\mathsf{P}^{\mathsf{BPP}} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$, and since the argument relativizes, the lemma follows. Suppose that $L \in \oplus\mathsf{P}^{\mathsf{BPP}}$. Then, there exists a language $A \in \mathsf{P}^{\mathsf{BPP}}$ such that

$$x \in L \iff |\{y : \langle x, y\rangle \in A\}| \text{ is odd.}$$

By lemma 7.5.3 we know that $\mathsf{P}^{\mathsf{BPP}} \subseteq \mathsf{BPP}^{\mathsf{BPP}} = \mathsf{BPP}$, so in fact $\mathsf{P}^{\mathsf{BPP}} = \mathsf{BPP}$, so we can assume directly that $A \in \mathsf{BPP}$. So we know that there exists a language $B \in \mathsf{P}$ such that

$$\langle x, y\rangle \in A \iff \langle x, y, z\rangle \in B,$$

and this is true for, say, a $(1 - 1/2^{p(|x|)})$-fraction of the z's, where p is any polynomial. We can make the polynomial p sufficiently large, so that in fact for a 3/4-fraction of the z's, we can make the following assertion:

$$\forall y[\langle x, y\rangle \in A \iff \langle x, y, z\rangle \in B].$$

We can make this assertion since the number of y's is itself bounded by $2^{q(|x|)}$, where q is a polynomial. It therefore follows that, for a given x, for a fraction of 3/4 of the z's, we have:

$$|\{y : \langle x, y\rangle \in A\}| = |\{y : \langle x, y, z\rangle \in B\}|$$

and in particular, the LHS is odd iff the RHS is odd. Thus, the following $\mathsf{BPP}^{\oplus\mathsf{P}}$ algorithm decides L: on input x_0, generate a random z_0, and then query the $\oplus\mathsf{P}$ oracle if the set $\{y|\langle x_0, y, z_0\rangle \in B\}$ is odd-sized. On 3/4 of the z's we get the right answer. □

Lemma 7.5.6 $\mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$.

PROOF: Suppose that $L \in \mathsf{PP}^{\oplus\mathsf{P}}$. Then, there exists a language $A \in \mathsf{P}^{\oplus\mathsf{P}}$, and hence, by lemma 7.5.3 (part two), $A \in \oplus\mathsf{P}$, such that:

$$x \in L \iff |\{y : \langle x, y\rangle \in A\}| > 2^{p(|x|)-1}.$$

Let M be the polytime nondeterministic TM witnessing that A is in $\oplus\mathsf{P}$. Using some clever arithmetic, it is possible to design a $\#\mathsf{P}$ function g, such that

$$g(\langle x, y\rangle) = \begin{cases} m2^{p(|x|)} + 1 & \text{if } \#\text{accept}_M(\langle x, y\rangle) \text{ is odd} \\ m2^{p(|x|)} & \text{if } \#\text{accept}_M(\langle x, y\rangle) \text{ is even,} \end{cases}$$

and from g we can easily design the #P function h,

$$h(x) = \sum_y g(\langle x, y\rangle).$$

The #P machine computing h simply guesses a y (in the appropriate range, of course), and then simulates $g(\langle x, y\rangle)$.

The question is how to define g.

Let $s_0(z) = z$, and let $s_{i+1}(z) = 3s_i(z)^4 + 4s_i(z)^3$. The function s_i has the property that if z is odd, then $s_i(z) - 1$ is divisible by 2^{2^i}, and if z is even, it is divisible by $m2^{2^i}$. On the other hand, we can evaluate a polynomial at a value in #P as follows: suppose we want to compute $a_0 + a_1 z + a_2 z^2 + \cdots + a_m z^m$. For each i, we branch out on all the values $1, 2, \ldots, a_i$, and then we branch out on the value z i-many times.

Let $l_x = \lceil \log(p(|x|)) + 1 \rceil$ and $r_x(z) = (s_{l_x}(z))^2$, and define $g(x) = r_x(\#\text{accept}_M(x))$.

We now show how to put it all together to decide L in $\mathsf{P}^{\#\mathsf{P}[1]}$. On input x, we use the #P oracle exactly once to compute $h(x)$. Once we have $h(x)$ on the oracle tape, we check whether the value encoded in the first $p(|x|)$ many bits is greater than $2^{p(|x|)-1}$, and accept iff it is. □

Lemma 7.5.7 $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$.

PROOF: Showing $\mathsf{P}^{\mathsf{PP}} \subseteq \mathsf{P}^{\#\mathsf{P}}$ is easy, since a #P oracle gives more information than a PP oracle (we not only know if the majority of computations are accepting; we actually know how many).

To show that $\mathsf{P}^{\mathsf{PP}} \supseteq \mathsf{P}^{\#\mathsf{P}}$, we show that for any polytime nondeterministic TM N, the language $L = \{\langle x, y\rangle | \#\text{accept}_N(x) \geq y\}$ is in PP. This way, if we have a #P function f as oracle, we can compute its value in polytime with a PP oracle using binary search.

We can assume that N is precise, and all computational paths are of length exactly $p(|x|)$. Now define the machine D, which takes input $\langle x, y\rangle$, and which initially branches on $b = 0$ and $b = 1$.

If $b = 0$, it guesses a string $z \in \{0,1\}^{p(|x|)}$, and then accepts iff the rank of z is at most $2^{p(|x|)} - y$. (The *rank* of a string is the number of strings, in lexicographic order by lengths, after that string. In this case it means that there are more than y many strings before z—of course, we work with the finite set $\{0,1\}^{p(|x|)}$.)

If $b = 1$, then D just simulates N on x.

The proportion of accepting computations is:

$$\frac{2^{p(|x|)} - y}{2^{p(|x|)+1}} + \frac{\#\text{accept}_N(x)}{2^{p(|x|)+1}},$$

and it is $> \frac{1}{2}$ iff $\#\text{accept}_N(x) \geq y$. □

7.6 Answers to selected exercises

Exercise 7.2.2. To prove that PP is closed under complementation, we need to show that it is possible to modify a nondeterministic TM in such a way, that the number of accepting and rejecting computation paths will always differ (of course preserving the inequality between them). To achieve this we first make sure that the left-most computation path is always an accepting one (this can be done preserving the accept/reject relation by splitting the computation tree into 4: an always-accepting, an always-rejecting and two identical to the original one). Then we add an additional bit to the state of the machine to keep track whether the computation follows the left-most path. If this is the case, we nondeterministically accept or reject. Otherwise we do as before (of course we need to create two identical branches to make the machine precise). Now if the original machine would accept on exactly half of the branches, the new one will accept on one less than a half. Therefore we can get a machine for the complement of the original language by reversing the accept and reject answers.

Knowing that PP is closed under complement it is easy to show that it is also closed under symmetric difference. That is because for any numbers p, q such that $\frac{1}{2} < p, q \leq 1$ either $p(1-q) + q(1-p)$ or $pq + (1-p)(1-q)$ has to be greater than $\frac{1}{2}$. Therefore for any two languages $L_1, L_2 \in \mathsf{PP}$ we can use one of the following to get their symmetric difference:

$$L_1 \triangle L_2 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$$
$$L_1 \triangle L_2 = \overline{(L_1 \cap L_2) \cup (\overline{L_1} \cap \overline{L_2})}$$

Exercise 7.2.3. We know that $\mathsf{RP} \subseteq \mathsf{NP}$. Assume that $\mathsf{NP} \subseteq \mathsf{BPP}$. Then $\textsc{Sat} \in \mathsf{BPP}$. Consider the following algorithm for SAT: given a formula ϕ we first use the BPP algorithm to see if it is satisfiable. If the answer comes "no", we reject. If the answer comes "yes", we set the first variable x_1 to 0 and 1 and ask the BPP machine again. If both answers are "no", we reject. Otherwise we proceed with x_1 set to the value for which the answer was "yes". We repeat this procedure until we get a total truth assignment. Then we verify (deterministically!) that it really satisfies ϕ and accept if and only if it does. From the last step it is clear, that our algorithm will not return false positive answers. The probability of a false negative can be bounded from above by the sum of probabilities of errors on each steps (and there are n of them). And as the probability of error of an BPP algorithm

can be made exponentially small by amplification, we can easily limit the resulting probability of false negatives by $\frac{1}{2}$. Thus SAT $\in$ RP, but as SAT is NP-complete, NP $\subseteq$ RP and finally, NP = RP.

Exercise 7.2.4. Consider any language $L \in$ PP. Having the machine solving it we can create a Boolean formula encoding possible computations (as in the Cook's Theorem 2.2.8). Let us consider two parts of this formula: ϕ, meaning "the computation is correct (well-formed)", and ψ, meaning "the computation is accepting". Let us introduce a new variable x and consider the following formula: $\theta := (\phi \supset \psi) \wedge (\neg\phi \supset x)$. It is easy to see, that if a truth assignment does not satisfy ϕ, then to satisfy θ it must satisfy x. It means that exactly half of these assignments satisfy θ. On the other hand if a truth assignment satisfies ϕ then, to satisfy θ, it must satisfy ψ. It means that more than half of all assignments satisfy θ if and only if more than a half correct computations are accepting. And, as the size of θ is of course polynomial in the size of the instance of L, the reduction works correctly. Thus any problem in PP can be reduced do MAJSAT which means that MAJSAT is PP-complete.

7.7 Notes

The material in section 7.1.2 is based on [Sip06, § 10.2]. The proof of Lemma 8.2.7 is from [Sip06, exercise 10.16]. Section 8.3 is based on [MR95, § 14.4, pg. 410]. Section 7.1.3 is based on [KR87]. Exercise 7.2.1 is [Pap94, exr. 11.5.13], exercise 7.2.2 is [Pap94, exr. 11.5.14 and 15], exercise 7.2.3 is [Pap94, exr. 11.5.18], exercise 7.2.4 is [Pap94, exr. 11.5.16]. The proof of lemma 7.3.1 is based on [Pap94, Lemma 11.9], and corollary 7.3.2 is based on [Pap94] as well, but note that in [Pap94] the condition $\varepsilon < \frac{1}{4}$ is omitted but it is necessary to ensure that $0 < \theta \leq 1$. Section 8.4 is based on [HO02, chapter 4].