6 Randomized algorithms

We cited Church's Thesis in Section 3: every "algorithm" (in the heuristic meaining of the word) is realizable on, e.g., a Turing machine. If in an algorithm, we permit "coin tossing" i.e., the generation of a random number as an elementary step, then provably, we will be able to solve problems the Turing machine cannot (we will show this in a later section). In other cases, the random choice significantly accelerates the algorithms. We will see examples for this in the present section.

Since in this way, we obtain a new, stronger mathematical notion of a machine, corresponding complexity classes can also be introduced. Some of the most important of these will be treated at the end of the section.

6.1 Verifying a polynomial identity

Let $f(x_1, \ldots, x_n)$ be a rational polynomial with n variables that has degree at most k in each of its variables. We would like to decide whether f is identically 0 (as a function of n variables). We know from classical algebra that a polynomial is identically 0 if and only if, after "opening its parentheses", all terms "cancel". This criterion is, however, not always useful. It is conceivable, e.g., that the polynomial is given in a parenthesized form an the opening of the parentheses leads to exponentially many terms as in

$$(x_1 + y_1)(x_2 + y_2) \cdots (x_n + y_n) + 1.$$

It would also be good to say something about polynomials in whose definition not only the basic algebraic operations occur but also some other ones, like the computation of a determinant (which is a polynomial itself but is often computed, as we have seen, in some special way).

The basic idea is that we write random numbers in place of the variables and compute the value of the polynomial. If this is not 0 then, naturally, the polynomial cannot be identically 0. If the computed value is 0 then though it can happen that the polynomial is not identically 0, but "hitting" one of its roots has very small probability; therefore in this case we can conclude that the polynomial is identically 0; the probability that we make a mistake is small.

If we could give real values to the variables, chosen according to the uniform distribution e.g. in the interval $[0, 1]$, then the probability of error would be 0. We must in reality, however, compute with discrete values; therefore we assume that the values of the variables are chosen from among the integers of the interval $[0, N-1]$, independently and according to the uniform distribution. In this case, the probability of error will not be 0 but it will be "small" if N is large enough. This is the meaning of the following fundamental result:

(6.1) **Schwartz's Lemma** *If f is a not identically 0 polynomial in n variables with degree at most k in each variable, and the values ξ_i $(i = 1, \ldots, n)$ are chosen in the interval $[0, N-1]$ independently of each other according to the uniform distribution then*

$$\text{Prob}\{f(\xi_1, \ldots, \xi_n) = 0\} \leq \frac{kn}{N}.$$

Proof We prove the assertion by mathematical induction on n. The statement is true for $n = 1$ since a polynomial of degree k can have at most k roots. Let $n > 1$ and let us arrange f according to the powers of x_1:

$$f = f_0 + f_1 x_1 + f_2 x_1^2 + \cdots + f_t x_1^t,$$

where $f_0, \ldots, f_t$ are polynomials of the variables $x_2, \ldots, x_n$, the term f_t is not identically 0, and $t \le k$. Now,

$$\begin{aligned}
&\mathrm{Prob}\{f(\xi_1,\ldots,\xi_n) = 0\} \\
&\le \ \mathrm{Prob}\{f(\xi_1,\ldots,\xi_n) = 0 \mid f_t(\xi_2,\ldots,\xi_n) = 0\}\mathrm{Prob}\{f_t(\xi_2,\ldots,\xi_n) = 0\} \\
&+ \ \mathrm{Prob}\{f(\xi_1,\ldots,\xi_n) = 0 \mid f_t(\xi_2,\ldots,\xi_n) \ne 0\}\mathrm{Prob}\{f_t(\xi_2,\ldots,\xi_n) \ne 0\} \\
&\le \ \mathrm{Prob}\{f_t(\xi_2,\ldots,\xi_n) = 0\} + \mathrm{Prob}\{f(\xi_1,\ldots,\xi_n) = 0 \mid f_t(\xi_2,\ldots,\xi_n) \ne 0\}.
\end{aligned}$$

Here, we can estimate the first term by the inductive assumption, and the second term is at most k/N (since ξ_1 is independent of the variables $\xi_2, \ldots, \xi_n$, therefore if the latter are fixed in such a way that $f_t \ne 0$ and therefore f as a polynomial of x_1 is not identically 0 then the probability that ξ_1 is its root is at most k/N). Hence

$$\mathrm{Prob}\{f(\xi_1,\ldots,\xi_n) = 0\} \le \frac{k(n-1)}{N} + \frac{k}{N} \le \frac{kn}{N}.$$

∎

This offers the following a **randomized** algorithm (i.e., one that uses randomness) to decide whether a polynomial f is identically 0:

(6.2) **Algorithm** *We compute $f(\xi_1,\ldots,\xi_n)$ with integer values ξ_i chosen randomly and independently of each other according to the uniform distribution in the interval $[0, 2kn]$. If we don't get the value 0 we stop: f is not identically 0. If we get 0 value we repeat the computation. If we get 0 value 100 times we stop and declare that f is identically 0.*

If f is identically 0 then this algorithm will determine this. If f is not identically 0 then in every separate iteration—according to Schwartz's Lemma—the probability that the result is 0 is less than 1/2. With 100 experiments repeated independently of each other, the probability that this occurs every time, i.e., that the algorithm asserts erroneously that f is identically 0, is less than 2^{-100}.

Two things are needed for us to be able to actually carry out this algorithm: on the one hand, we must be able to generate random numbers (here, we assume this can be implemented, and even in time polynomial in the number of bits of the integers to be generated), on the other hand, we must be able to evaluate f in polynomial time (the size of the input is the length of the "definition" of f; this definition can be, e.g., an expression containing multiplications and additions with parentheses, but also something entirely different, e.g., a determinant form).

As a surprising example for the application of the method we present a matching algorithm. (We have already treated the matching problem in Subsection 4.1). Let G be a bipartite graph with the edge set $E(G)$ whose edges run between sets A and B, $A = \{a_1, \ldots, a_n\}$,

$B = \{b_1, \ldots, b_n\}$. Let us assign to each edge $a_i b_j$ a variable x_{ij}. Let us construct the $n \times n$ matrix M as follows:

$$m_{ij} = \begin{cases} x_{ij} & \text{if } a_i b_j \in E(G), \\ 0 & \text{otherwise.} \end{cases}$$

The determinant of this graph is closely connected with the matchings of the graph G as Dénes Kőnig noticed while analyzing a work of Frobenius (compare with Theorem 5.9):

(6.3) **Theorem** *There is a complete matching in the bipartite graph G if and only if* $\det(M)$ *is not identically 0.*

Proof Consider a term in the expansion of the determinant:

$$\pm m_{1\pi(1)} m_{2\pi(2)} \cdots m_{n\pi(n)},$$

where π is a permutation of the numbers $1, \ldots, n$. For this not to be 0, we need that a_i and $b_{\pi(i)}$ be connected for all i; in other words, that $\{a_1 b_{\pi(1)}, \ldots, a_n b_{\pi(n)}\}$ be a complete matching in G. In this way, if there is no complete matching in G then the determinant is identically 0. If there are complete matchings in G then to each one of them a nonzero expansion term corresponds. Since these terms do not cancel each other (any two of them contain at least two different variables), the determinant is not identically 0. ∎

Since $\det(M)$ is a polynomial of the elements of the matrix M that is computable in polynomial time (e.g. by Gaussian elimination) this theorem offers a polynomial-time randomized algorithm for the matching problem in bipartite graphs. We mentioned it before that there is also a polynomial-time deterministic algorithm for this problem (the "Hungarian method"). One advantage of the algorithm treated here is that it is very easy to program (determinant-computation can generally be found in the program library). If we use "fast" matrix multiplication methods then this randomized algorithm is a little faster than the fastest known deterministic one: it can be completed in time $O(n^{2.4})$ instead of $O(n^{2.5})$. Its main advantage is, however, that it is well suitable to parallelization, as we will see in a later section.

In non-bipartite graphs, it can also be decided by a similar but slightly more complicated method whether there is a complete matching. Let $V = \{v_1, \ldots, v_n\}$ be the vertex set of the graph G. Assign again to each edge $v_i v_j$ (where $i < j$) a variable x_{ij} and construct an asymmetric $n \times n$ matrix $T = (t_{ij})$ as follows:

$$t_{ij} = \begin{cases} x_{ij} & \text{if } v_i v_j \in E(G) \text{ and } i < j, \\ -x_{ij} & \text{if } v_i v_j \in E(G) \text{ and } i > j, \\ 0 & \text{otherwise.} \end{cases}$$

The following analogue of the above cited Frobenius-Kőnig theorem comes from Tutte and we formulate it here without proof:

(6.4) **Theorem** *t:MatchingDet There is a complete matching in the graph G if and only if* $\det(T)$ *is not identically 0.*

This theorem offers, similarly to the case of the bipartite graph, a randomized algorithm for deciding whether there is a complete matching in G.

6.1 **Exercise** Suppose that some experiment has some probability p of success. Prove that in n^3 experiments, it is possible to compute an approximation $\hat{p}$ of p such that the probability of $|p - \hat{p}| > \sqrt{p(1-p)}/n$ is at most $1/n$. [Hint: Use Tshebysheff's Inequality.] $\diamondsuit$

6.2 **Exercise** Suppose that somebody gives you three $n \times n$ matrices A, B, C (of integers of maximimum length l) and claims $C = AB$. You are too busy to verify this claim exactly and do the following. You choose a random vector x of length n whose entries are integers chosen uniformly from some interval $[0, \dots, N-1]$, and check $A(Bx) = Cx$. If this is true you accept the claim otherwise you reject it.

(a) How large must N be chosen to make the probability of false acceptance smaller than 0.01?

(b) Compare the time complexity the probabilistic algorithm to the one of the deterministic algorithm computing AB.

$\diamondsuit$

6.2 Prime testing

Let m be an odd natural number, we want to decide whether it is a prime. We have seen in the previous chapter that this problem is in NP $\cap$ co$-$NP. The witnesses described there did not lead, however (at least for the time being) to a polynomial-time prime test. We will therefore give first a new, more complicated NP description of compositeness.

(6.1) **"Little" Fermat Theorem** *If m is a prime then $a^{m-1} - 1$ is divisible by m for all natural numbers $1 \le a \le m-1$.*

If—with given m—the integer $a^{m-1} - 1$ is divisible by m then we say that a **satisfies the Fermat condition**.

The Fermat condition, when required for all integers $1 \le a \le m-1$, also characterizes the primes since if m is a composite number, and we choose for a any integer not relatively prime to m, then $a^{m-1} - 1$ is obviously not divisible by m. Of course, we cannot check the Fermat condition for every a: this would take exponential time. The question is therefore to which a's should we apply it? There are composite numbers m (the so-called pseudo-primes) for which the Fermat condition is satisfied for all primitive residue classes a; for such numbers, it will be especially difficult to find an integer a violating the condition. (Such pseudo-prime is e.g. $561 = 3 \cdot 11 \cdot 17$.)

Let us recall that the set of integers giving identical remainder after division by a number m is called a **residue class** modulo m. This residue class is **primitive** if its members are relatively prime to m (this is satisfied obviously at the same time for all elements of the residue class, just as the Fermat condition). In what follows the residue classes that satisfy the Fermat condition (and are therefore necessarily relatively prime to m) will be called **Fermat-accomplices**. The residue classes violating the condition are, on the other hand, the **traitors**.

(6.2) **Lemma** *If m is not a pseudo-prime then at most half of the modulo m primitive residue classes is a Fermat-accomplice.*

Note that none of the non-primitive residue classes are Fermat accomplices.

Proof If we multiply all accomplices by a traitor relatively prime to m then we get all different traitors. ∎

Thus, if m is not a pseudo-prime then the following randomized prime test works: check whether a randomly chosen integer $1 \le a \le m-1$ satisfies the Fermat condition. If not then we know that m is not a prime. If yes then repeat the procedure. If we found 100 times, independently of each other, that the Fermat condition is satisfied then we say that m is a prime. It can though happen that m is composite but if m is not a pseudo prime then the probability to have found an integer a satisfying the condition was less than $1/2$ in every step, and the probability that this should occur 100 consecutive times is less than 2^{-100}.

Unfortunately, this method fails for pseudoprimes (it finds them prime with large probability). We will therefore modify the Fermat condition somewhat. Let us write the number $m-1$ in the form $2^k M$ where M is odd. We say that M **satisfies the Miller condition** if at least one of the numbers

$$a^M - 1, a^M + 1, a^{2M} + 1, a^{4M} + 1, \ldots, a^{2^{k-1}M} + 1$$

is divisible by m. Since the product of these numbers is just $a^{m-1}-1$, every number satisfying the Miller condition also satisfies the Fermat condition (but not conversely, since m can be composite and thus can be a divisor of a product without being the divisor of any of its factors).

(6.3) **Lemma** *m is a prime if and only if every integer $1 \le a \le m-1$ satisfies the Miller condition.*

Proof

1. If m is composite then each of its proper divisors violates the Miller condition.

2. Suppose that m is prime. Then according to the Fermat condition, $a^{m-1} - 1$ is divisible by m for every integer $1 < a < m$. This number can be decomposed, however, into the product

$$a^{m-1} - 1 = (a^M - 1)(a^M + 1)(a^{2M} + 1)(a^{4M} + 1) \cdots (a^{2^{k-1}M} + 1).$$

Hence (using again the fact that m is a prime) one of these factors must also be divisible by m, i.e. a satisfies the Miller condition. ∎

We will need some fundamental remarks on divisibility in general and on pseudoprimes in particular.

(6.4) **Lemma** *Every pseudoprime m is*

(a) *Odd*

(b) *Squarefree (is not divisible by any square).*

Proof

(a) If $m > 2$ is even then $a = -1$ (if somebody prefers positive remainders they can take $m-1$) will be a Fermat traitor since $(-1)^{m-1} \equiv -1 \not\equiv 1 \pmod{m}$.

(b) Assume that $p^2 \mid m$; let k be the largest exponent for which $p^k \mid m$. Then $a = m/p - 1$ is a Fermat traitor since the last two terms of the binomial expansion of $(m/p-1)^{m-1}$ are $-(m-1)(m/p) + 1 \equiv m/p + 1 \not\equiv 1 \pmod{p^k}$ (all earlier terms are divisible by p^k) and if an integer is not divisible by p^k then it is not divisible by m either. ∎

(6.5) **Lemma** *Let $m = p_1 p_2 \cdots p_t$ where the p_i's are different primes. The relation $a^k \equiv 1 \pmod{m}$ holds for all a relatively prime to m if and only if $p_i - 1$ divides k for all i with $1 \le i \le t$.*

Proof If $p_i - 1$ divides k for all i with $1 \le i \le t$ then $a^k - 1$ is divisible by p_i according to the little Fermat Theorem and then it is also divisible by m. Conversely, suppose that $a^k \equiv 1 \pmod{m}$ for all a relatively prime to m. If e.g. $p_1 - 1$ would not divide k then let g be a primitive root modulo p_1 (the existence of primitive roots was spelled out in Theorem 5.16). According to the Chinese Remainder Theorem, there is a residue class h modulo m with $h \equiv g \pmod{p_1}$ and $h \equiv 1 \pmod{p_i}$ for all $i \ge 2$. Then $(h, m) = 1$ and $p_1 \not| h^k - 1$, so $m \not| h^k - 1$. ∎

(6.6) **Corollary** *The number m is a pseudoprime if and only if $m = p_1 p_2 \cdots p_t$ where the p_i's are different primes, $t \ge 2$ and $(p_i - 1)$ divides $(m-1)$ for all i with $1 \le i \le t$.*

(6.7) **Remark** This is how one can show about the above example 561 that it is a pseudoprime. ♢

The key idea of the algorithm is the result that in case of a composite number—contrary to the Fermat condition—the majority of the residue classes violates the Miller condition.

(6.8) **Theorem** *If m is a composite number then at least half of the primitive residue classes modulo m violate the Miller condition* (we can call these Miller traitors).

Proof Since we have already seen the truth of the lemma for non-pseudoprimes, in what follows we can assume that m is a pseudoprime. Let $p_1 \cdots p_t$ ($t \ge 2$) be the prime decomposition of m. According to the above Corollary, we have $(p_i - 1) \mid (m-1) = 2^k M$ for all i with $1 \le i \le t$.

Let l be the largest exponent with the property that none of the numbers $p_i - 1$ divides $2^l M$. Since the numbers $p_i - 1$ are even while M is odd, such an exponent exists (e.g. 0) and clearly $0 \le l < k$. Further, by the definition of l, there is a j for which $p_j - 1$ divides $2^{l+1} M$. Therefore $p_j - 1$ divides $2^s M$ for all s with $l < s \le k$, and hence p_j divides $a^{2^s M} - 1$ for all primitive residue classes a. Consequently p_j cannot divide $a^{2^s M} + 1$ which is larger by 2, and hence m does not divide $a^{2^s M} + 1$ either. If therefore a is a residue class that is a Miller accomplice then m must already be a divisor of one of the remainder classes $a^M - 1, a^M + 1,$

$a^{2M}+1, \ldots, a^{2^{l-1}M}+1, a^{2^l M}+1$. Hence for each such a, the number m divides either the product of the first $l+1$, which is $(a^{2^l M}-1)$, or the last one, $(a^{2^l M}+1)$. Let us call the primitive residue class a modulo m an "accomplice of the first kind" if $a^{2^l M} \equiv 1 \pmod{m}$ and an "accomplice of the second kind" if $a^{2^l M} \equiv -1 \pmod{m}$.

Let us estimate first the number of accomplices of the first kind. Consider an index i with $1 \le i \le t$. Since $p_i - 1$ does not divide the exponent $2^l M$, Lemma 6.5 implies that there is a number a not divisible by p_i for which $a^{2^l M} - 1$ is not divisible by p_i. (This is actually a Fermat traitor belonging to the exponent $2^l M$, mod p_i—of course, not mod m!) The reasoning of Lemma 6.2 shows that then at most half of the mod p_i residue classes will be Fermat accomplices belonging to the above exponent, i.e. such that $a^{2^l M} - 1$ is divisible by p_i. According to the Chinese Remainder Theorem, there is a one-to-one correspondence between the primitive residue classes with respect to the product $p_1 \cdots p_t$ as modulus and the t-tuples of primitive residue classes modulo the primes $p_1, \ldots, p_t$. Thus, modulo $p_1 \cdots p_t$, at most a 2^t-th part of the primitive residue classes is such that every p_i divides $(a^{2^l M} - 1)$. Therefore at most a 2^t-th part of the mod m primitive residue classes are accomplices of the first kind.

It is easy to see that the product of two accomplices of the second kind is one of the first kind. Hence multiplying all accomplices of the second kind by a fixed one of the second kind, we obtain accomplices of the first kind, and thus the number of accomplices of the second kind is at least as large as the number of accomplices of the first kind. (If there is no accomplice of the second kind to multiply with then the situation is even better: zero is certainly not greater than the number of accomplices of the first kind.) Hence even the two kinds together make up at most a 2^{t-1}-th part of the primitive residue classes, and so (due to $t \ge 2$) at most a half. ∎

(6.9) **Lemma** *For a given m and a, it is decidable in polynomial time whether a satisfies the Miller condition.*

For this, it is enough to recall Lemma 4.3: the remainder of a^b modulo c is computable in polynomial time. Based on these three lemmas, the following randomized algorithm can be given for prime testing:

(6.10) **Algorithm** We choose a number between 1 and $m-1$ randomly and check whether it satisfies the Miller condition. If it does not then m is composite. If it does then we choose a new a. If the Rabin condition is satisfied 100 times consecutively then we declare that m is a prime. $\diamondsuit$

If m is a prime then the algorithm will certainly assert this. If m is composite then the number a chosen randomly violates the Rabin condition with probability 1/2. After hundred independent experiments the probability will therefore be at most 2^{-100} that the Rabin condition is not violated even once, i.e., that the algorithm asserts that m is a prime.

(6.11) **Remarks**

1. If m is found composite by the algorithm then—interestingly—we see this not from its finding a divisor but (essentially) from the fact that one of the residues violates the

Miller condition. If at the same time, the residue a does not violate the Fermat condition then m cannot be relatively prime to each of the numbers $a^M - 1$, $a^M + 1$, $a^{2M} + 1$, $a^{4m} + 1$, $\cdots$, $a^{2^{k-1}M} + 1$, therefore computing its greatest common divisors with each, one of them will be a proper divisor of m. No polynomial algorithm (either deterministic or randomized) is known for finding a polynomial algorithm in the case when the Fermat condition is also violated. This problem is significantly more difficult also in practice than the decision of primality. We will see in the section on cryptography that this empirical fact has important applications.

2. For a given m, we can try to find an integer a violating the Rabin condition not by random choice but by trying out the numbers 1,2, etc. It is not known how small is the first such integer if m is composite. Using, however, a hundred year old conjecture of analytic number theory, the so-called Generalized Riemann Hypothesis, one can show that it is not greater than $\log_2 m$. Thus, this deterministic prime test works in polynomial time if the Generalized Riemann Hypothesis is true.

$\diamondsuit$

We can use the prime testing algorithm learned above to look for a prime number with n digits (say, in the binary number system). Choose namely a number k randomly from the interval $[2^{n-1}, 2^n - 1]$ and check whether it is a prime, say, with an error probability of at most $2^{-100}/n$. If it is, we stop. If it is not we choose a new number k. Now, it follows from the theory of prime numbers that in this interval, not only there is a prime number but the number of primes is rather large: asymptotically $(\log e)2^{n-1}/n$, i.e., a randomly chosen n-digit number will be a prime with probability cca. $(\log e)/n$. Repeating therefore this experiment $O(n)$ times we find a prime number with very large probability.

We can choose a random prime similarly from any sufficiently long interval, e.g. from the interval $[1, 2^n]$.

6.3 **Exercise** Show that if m is a pseudoprime then the above algorithm not only discovers this with large probability but it can also be used to find a decomposition of m into two factors. $\diamondsuit$

6.3 Randomized complexity classes

In the previous subsections, we treated algorithms that used random numbers. Now we define a class of problems solvable by such algorithms.

First we define the corresponding machine. Let $T = (k, \Sigma, \Gamma, \Phi)$ be a non-deterministic Turing machine and let us be given a probability distribution for every $g \in \Gamma$, $h_1, \ldots, h_k \in \Sigma$ on the set

$$\{ (g', h'_1, \ldots, h'_k, \varepsilon_1, \ldots, \varepsilon_k) : (g, h_1, \ldots, h_k, g', h'_1, \ldots, h'_k, \varepsilon_1, \ldots, \varepsilon_k) \in \Phi \}.$$

(It is useful to assume that the probabilities of events are rational numbers, since then events with such probabilities are easy to generate, provided that we can generate mutually independent bits.) A non-deterministic Turing machine together with these distributions is called a **randomized Turing machine**.

Every legal computation of a randomized Turing machine has some probability. We say that a randomized Turing machine **weakly decides** (or, **decides in the Monte-Carlo sense**) a language $\mathcal{L}$ if for all inputs $x \in \Sigma^*$, it stops with probability at least 3/4 in such a way that in case of $x \in \mathcal{L}$ it writes 1 on the result tape, and in case of $x \notin \mathcal{L}$, it writes 0 on the result tape. Shortly: the probability that it gives a wrong answer is at most 1/4.

In our examples, we used randomized algorithms in a stronger sense: they could err only in one direction. We say that a randomized Turing machine **accepts** a language $\mathcal{L}$ if for all inputs x, it always rejects the word x in case of $x \notin \mathcal{L}$, and if $x \in \mathcal{L}$ then the probability is at least 1/2 that it accepts the word x.

We say that a randomized Turing machine **strongly decides** (or, **decides in the Las Vegas sense**) a language $\mathcal{L}$ if it gives a correct answer for each word $x \in \Sigma^*$ with probability 1. (Every single computation of finite length has positive probability and so the 0-probability exception cannot be that the machine stops with a wrong answer, only that it works for an infinite time.)

In case of a randomized Turing machine, for each input, we can distunguish the number of steps in the longest computation and the expected number of steps. The class of all languages that are weakly decidable on a randomized Turing machine in polynomial expected time is denoted by BPP (Bounded Probability Polynomial). The class of languages that can be accepted on a randomized Turing machine in polynomial expected time is denoted by RP (Random Polynomial). The class of all languages that can be strongly decided on a randomized Turing machine in polynomial expected time is denoted by ΔRP. Obviously, BPP $\supset$ RP $\supset$ ΔRP $\supset$ P.

The constant 3/4 in the definition of weak decidability is arbitrary: we could say here any number smaller than 1 but greater than 1/2 without changing the definition of the class BPP (it cannot be 1/2: with this probability, we can give a correct answer by coin-tossing). If namely the machine gives a correct answer with probability $1/2 < c < 1$ then let us repeat the computation t times on input x and accept as answer the one given more times. It is easy to see from the Law of Large Numbers that the probability that this answer is wrong is less than c_1^t where c_1 is a constant smaller than 1 depending only on c. For sufficiently large t this can be made arbitrarily small and this changes the expected number of steps only by a constant factor.

It can be similarly seen that the constant 1/2 in the definition of acceptance can be replaced with an arbitrary positive number smaller than 1.

Finally, we note that instead of the expected number of steps in the definition of the classes BPP and RP, we could also consider the largest number of steps; this would still not change the classes. Obviously, if the largest number of steps is polynomial, then so is the expected number of steps. Conversely, if the expected number of steps is polynomial, say, at most $|x|^d$, then according to Markov's Inequality, the probability that a computation lasts a longer time than $8|x|^d$ is at most 1/8. We can therefore build in a counter that stops the machine after $8|x|^d$ steps, and writes 0 on the result tape. This increases the probability of error by at most 1/8.

The same is, however, not known for the class ΔRP: the restriction of the longest running time would lead here already to a deterministic algorithm, and it is not known whether ΔRP is equal to P (moreover, this is rather expected not to be the case; there are examples for problems solvable by polynomial Las Vegas algorithms for which no polynomial deterministic algorithm is known).

(6.1) **Remark** A Turing machine using randomness could also be defined in a different way: we could consider a deterministic Turing machine which has, besides the usual (input-, work- and result-) tapes also a tape on whose every cell a bit (say, 0 or 1) is written that is selected randomly with probability 1/2. The bits written on the different cells are mutually independent. The machine itself works deterministically but its computation depends, of course, on chance (on the symbols written on the random tape). It is easy to see that such a deterministic Turing machine fitted with a random tape and the non-deterministic Turing machine fitted with a probability distribution can replace each other in all definitions.

We could also define a randomized Random Access Machine: this would have an extra cell w in which there is always a 0 or 1 with probability 1/2. We have to add the instruction $y := w$ to the programming language. Every time this is executed a new random bit occurs in the cell w that is completely independent of the previous bits. Again, it is not difficult to see that this does not bring any significant difference. $\diamondsuit$

6.4 **Exercise** Show that the Turing machine equipped with a random tape and the non-deterministic Turing machine equipped with a probability distribution are equivalent: if some language is accepted in polynomial time by the one then it is also accepted by the other one. $\diamondsuit$

6.5 **Exercise** Formulate what it means that a randomized RAM accepts a certain language in polynomial time and show that this is equivalent to the fact that some randomized Turing machine accepts it. $\diamondsuit$

It can be seen that every language in RP is also in NP. It is trivial that the classes BPP and ΔRP are closed with respect to the taking of complement: they contain, together with every language $\mathcal{L}$ the language $\Sigma^* \setminus \mathcal{L}$. The definition of the class RP is not such and it is not known whether this class is closed with respect to complement. It is therefore worth defining the class co$-$RP: A language $\mathcal{L}$ is in co$-$RP if $\Sigma^* \setminus \mathcal{L}$ is in RP.

"Witnesses" gave a useful characterization of the class NP. An analogous theorem holds also for the class RP.

(6.2) **Theorem** *A language $\mathcal{L}$ is in* RP *if and only if there is a language $\mathcal{L}' \in$* P *and a polynomial $f(n)$ such that*

(a) $\mathcal{L} = \{\, x \in \Sigma^* : y \in \Sigma^{f(|x|)}\ x\&y \in \mathcal{L}' \,\}$ *and*

(b) *if $x \in \mathcal{L}$ then at least half of the words y of length $f(|x|)$ are such that $x\&y \in \mathcal{L}'$.*

Proof Similar to the proof of the corresponding theorem on NP. ∎

The connection of the classes RP and ΔRP is closer than it could be expected on the basis of the analogy to the classes NP and P:

(6.3) **Theorem** *The following properties are equivalent for a language $\mathcal{L}$:*

(i) $\mathcal{L} \in \Delta$RP;

(ii) $\mathcal{L} \in RP \cap \text{co}-\text{RP}$;

(iii) *There is a randomized Turing machine with polynomial (worst-case) running time that can write, besides the symbols "0" and "1", also the words "I GIVE UP"; the answers "0" and "1" are never wrong, i.e., in case of $x \in \mathcal{L}$ the result is "1" or "I GIVE UP", and in case of $x \notin \mathcal{L}$ it is "0" or "I GIVE UP". The probability of the answer "I GIVE UP" is at most 1/2.*

Proof It is obvious that (i) implies (ii). It can also be easily seen that (ii) implies (iii). Let us submit x to a randomized Turing machine that accepts $\mathcal{L}$ in polynomial time and also to one that accepts $\Sigma^* \setminus \mathcal{L}$ in polynomial time. If the two give opposite answers then the answer of the first machine is correct. If they give identical answers then we "give it up". In this case, one of them made an error and therefore this has a probability at most 1/2.

Finally, to see that (iii) implies (i) we just have to modify the Turing machine T_0 given in (iii) in such a way that instead of the answer "I GIVE IT UP", it should start again. If on input x, the number of steps of T_0 is τ and the probability of giving it up is p then on this same input, the expected number of steps of the modified machine is

$$\sum_{t=1}^{\infty} p^{t-1}(1-p)t\tau = \frac{\tau}{1-p} \leq 2\tau.$$

■

We have seen in the previous subsection that the "language" of composite numbers is in RP. Even more is true: lately, Adleman and Huang have shown that this language is also in ΔRP. For our other important example, the not identically 0 polynomials, it is only known that they are in RP. Among the algebraic (mainly group-theoretical) problems, there are many that are in RP or ΔRP but no polynomial algorithm is known for their solution.

(6.4) **Remark** The algorithms that use randomization should not be confused with the algorithms whose performance (e.g., the expected value of their number of steps) is being examined for random inputs. Here we did not assume any probability distribution on the set of inputs, but considered the worst case. The investigation of the behavior of algorithms on random inputs coming from a certain distribution is an important but difficult area, still in its beginnings, that we will not treat here. ♢

6.6 **Exercise** Let us call a Boolean formula with n variables *simple* if it is either unsatisfiable or has at least $2^n/n^2$ satisfying assignments. Give a probabilistic polynomial algorithm to decide the satisfiability of simple formulas. ♢