Notes for Lecture 8

In this lecture we will define the probabilistic complexity classes **BPP**, **RP**, **ZPP** and we will see how they are related to each other, as well as to other deterministic or circuit complexity classes. Then we will present a randomized reduction showing that the SAT problem remains as hard when restricted to inputs that have either zero or one satisfying assignments as in the general case.

1 Probabilistic complexity classes

First we are going to describe the probabilistic model of computation. In this model an algorithm A gets as input a sequence of random bits r and the "real" input x of the problem. The output of the algorithm is the correct answer for the input x with some probability.

Definition 1 *An algorithm A is called a polynomial time probabilistic algorithm if the size of the random sequence $|r|$ is polynomial in the input $|x|$ and $A()$ runs in time polynomial in $|x|$.*

If we want to talk about the correctness of the algorithm, then informally we could say that for every input x we need $\mathbf{Pr}[A(x,r) = \text{correct answer for } x] \geq \frac{2}{3}$. That is, for every input the probability distribution over all the random sequences must be some constant bounded away from $\frac{1}{2}$. Let us now define the class **BPP**.

Definition 2 *A decision problem L is in* **BPP** *if there is a polynomial time algorithm A and a polynomial $p()$ such that :*

$$\forall x \in L \quad \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] \geq 2/3$$

$$\forall x \notin L \quad \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] \leq 1/3$$

We can see that in this setting we have an algorithm with two inputs and some constraints on the probabilities of the outcome. In the same way we can also define the class **P** as:

Definition 3 *A decision problem L is in* **P** *if there is a polynomial time algorithm A and a polynomial $p()$ such that :*

$$\forall x \in L \quad : \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] = 1$$

$$\forall x \notin L : \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] = 0$$

Similarly, we define the classes **RP** and **ZPP**.

Definition 4 *A decision problem L is in* **RP** *if there is a polynomial time algorithm A and a polynomial $p()$ such that:*

$$\forall x \in L \quad \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] \geq 1/2$$

$$\forall x \notin L \quad \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] \leq 0$$

Definition 5 *A decision problem L is in* **ZPP** *if there is a polynomial time algorithm A whose output can be $0, 1, ?$ and a polynomial $p()$ such that :*

$$\forall x \quad \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) =?] \leq 1/2$$
$$\forall x, \forall r \quad \textit{such that} \quad A(x,r) \neq? \quad \textit{then} \quad A(x,r) = 1 \quad \textit{if and only if} \quad x \in L$$

2 Relations between complexity classes

After defining these probabilistic complexity classes, let us see how they are related to other complexity classes and with each other.

Theorem 1 RP⊆NP.

PROOF: Suppose we have a **RP** algorithm for a language L. Then this algorithm is can be seen as a "verifier" showing that L is in **NP**. If $x \in L$ then there is a random sequence r, for which the algorithm answers yes, and we think of such sequences r as witnesses that $x \in L$. If $x \notin L$ then there is no witness. □

We can also show that the class **ZPP** is no larger than **RP**.

Theorem 2 ZPP⊆RP.

PROOF: We are going to convert a **ZPP** algorithm into an **RP** algorithm. The construction consists of running the **ZPP** algorithm and anytime it outputs ?, the new algorithm will answer 0. In this way, if the right answer is 0, then the algorithm will answer 0 with probability 1. On the other hand, when the right answer is 1, then the algorithm will give the wrong answer with probability less than $1/2$, since the probability of the **ZPP** algorithm giving the output ? is less than $1/2$. □

Another interesting property of the class **ZPP** is that it's equivalent to the class of languages for which there is an average polynomial time algorithm that always gives the right answer. More formally,

Theorem 3 *A language L is in the class* **ZPP** *if and only if L has an average polynomial time algorithm that always gives the right answer.*

PROOF: First let us clarify what we mean by average time. For each input x we take the average time of $A(x, r)$ over all random sequences r. Then for size n we take the worst time over all possible inputs x of size $|x| = n$. In order to construct an algorithm that always gives the right answer we run the **ZPP** algorithm and if it outputs a ?, then we run it again. Suppose that the running time of the **ZPP** algorithm is T, then the average running time of the new algorithm is:

$$T_{avg} = \frac{1}{2} \cdot T + \frac{1}{4} \cdot 2T + \ldots + \frac{1}{2^k} \cdot kT = O(T)$$

Now, we want to prove that if the language L has an algorithm that runs in polynomial average time $t(|x|)$, then this is in **ZPP**. We run the algorithm for time $2t(|x|)$ and output a ? if the algorithm has not yet stopped. It is straightforward to see that this belongs to

ZPP. First of all, the worst running time is polynomial, actually $2t(|x|)$. Moreover, the probability that our algorithm outputs a ? is less than $1/2$, since the original algorithm has an average running time $t(|x|)$ and so it must stop before time $2t(|x|)$ at least half of the times. □

Let us now prove the fact that **RP** is contained in **BPP**.

Theorem 4 RP⊆BPP

PROOF: We will convert an **RP** algorithm into a **BPP** algorithm. In the case that the input x does not belong to the language then the **RP** algorithm always gives the right answer, so it certainly satisfies that **BPP** requirement of giving the right answer with probability at least $2/3$. In the case that the input x does belong to the language then we need to improve the probability of a correct answer from at least $1/2$ to at least $2/3$.

Let A be an **RP** algorithm for a decision problem L. We fix some number k and define the following algorithm:

$A^{(k)}$

input: x,
  pick $r_1, r_2, \ldots, r_k$
  if $A(x, r_1) = A(x, r_2) = \ldots = A(x, r_k) = 0$ **then return** 0
  else return 1

Let us now consider the correctness of the algorithm. In case the correct answer is 0 the output is always right. In the case where the right answer is 1 the output is right except when all $A(x, r_i) = 0$.

$$\text{if} \quad x \notin L \quad \mathbf{Pr}_{r_1,\ldots,r_k}[A^k(x, r_1, \ldots, r_k) = 1] = 0$$

$$\text{if} \quad x \in L \quad \mathbf{Pr}_{r_1,\ldots,r_k}[A^k(x, r_1, \ldots, r_k) = 1] \geq 1 - \left(\frac{1}{2}\right)^k$$

It is easy to see that by choosing an appropriate k the second probability can go arbitrarily close to 1. In particular, choosing $k = 2$ suffices to have a probability larger than $2/3$, which is what is required by the definition of **BPP**. In fact, by choosing k to be a polynomial in $|x|$, we can make the probability *exponentially* close to 1. This means that the definition of **RP** that we gave above would have been equivalent to a definition in which, instead of the bound of $1/2$ for the probability of a correct answer when the input is in the language L, we had have a bound of $1 - \left(\frac{1}{2}\right)^{q(|x|)}$, for a fixed polynomial q. □

Let, now, A be a **BPP** algorithm for a decision problem L. Then, we fix k and define the following algorithm:

$A^{(k)}$

input: x,
  pick $r_1, r_2, \ldots, r_k$
  $c = \sum_{i=1}^{k} A(x, r_i)$
  if $c \geq \frac{k}{2}$ **then return** 1
  else return 0

In a **BPP** algorithm we expect the right answer to come up with probability more than 1/2. So, by running the algorithm many times we make sure that this slightly bigger than 1/2 probability will actually show up in the results. More formally let us define the Chernoff bounds.

Theorem 5 *(Chernoff Bound)*
Suppose $X_1, \ldots, X_k$ are independent random variables with values in $\{0, 1\}$ and for every i, $\mathbf{Pr}[X_i = 1] = p$. *Then:*

$$\mathbf{Pr}[\tfrac{1}{k}\sum_{i=1}^{k} X_i - p > \epsilon] < e^{-\epsilon^2 \frac{k}{2p(1-p)}}$$

$$\mathbf{Pr}[\tfrac{1}{k}\sum_{i=1}^{k} X_i - p < -\epsilon] < e^{-\epsilon^2 \frac{k}{2p(1-p)}}$$

The Chernoff bounds will enable us to bound the probability that our result is far from the expected. Indeed, these bounds say that this probability is exponentially small in respect to k.

Let us now consider how the Chernoff bounds apply to the algorithm we described previously. We fix the input x and call $p = \mathbf{Pr}_r[A(x, r) = 1]$ over all possible random sequences. We also define the independent random variables $X_1, \ldots, X_k$ such that $X_i = A(x, r_i)$.

First, suppose $x \in L$. Then the algorithm $A^{(k)}(x, r_1, \ldots, r_k)$ outputs the right answer 1, when $\frac{1}{k}\sum_i X_i \geq \frac{1}{2}$. So, the algorithm makes a mistake when $\frac{1}{k}\sum_i X_i < \frac{1}{2}$.

We now apply the Chernoff bounds to bound this probability.

$$\begin{aligned}\mathbf{Pr}[A^{(k)}\text{outputs the wrong answer on } \ x] &= \mathbf{Pr}[\tfrac{1}{k}\sum_i X_i < \tfrac{1}{2}] \\ &\leq \mathbf{Pr}[\tfrac{1}{k}\sum_i X_i - p \leq -\tfrac{1}{6}]\end{aligned}$$

since $p \geq \frac{2}{3}$.

$$\leq e^{-\frac{k}{72p(1-p)}} = 2^{-\Omega(k)}$$

The probability is exponentially small in k. The same reasoning applies also for the case where $x \notin L$. Further, it is easy to see that by choosing k to be a polynomial in $|x|$ instead

of a constant, we can change the definition of a **BPP** algorithm and instead of the bound of $\frac{1}{3}$ for the probability of a wrong answer, we can have a bound of $2^{-q(|x|)}$, for a fixed polynomial q.

Next, we are going to see how the probabilistic complexity classes relate to circuit complexity classes and specifically prove that the class **BPP** has polynomial size circuits.

Theorem 6 (Adleman) $\mathbf{BPP} \subseteq \mathbf{SIZE}(n^{O(1)})$

PROOF: Let L be in the class **BPP**. Then by definition, there is a polynomial time algorithm A and a polynomial p, such that for every input x

$$\mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = \text{wrong answer for} \quad x] \leq 2^{-(n+1)}$$

This follows from our previous conclusion that we can replace $\frac{1}{3}$ with $2^{-q(|x|)}$. We now fix n and try to construct a family of circuits C_n, that solves L on inputs of length n.

Claim 7 *There is a random sequence $r \in \{0,1\}^{p(n)}$ such that for every $x \in \{0,1\}^n$ $A(x,r)$ is correct.*

PROOF: Informally, we can see that for each input x the number of random sequences r that give the wrong answer is exponentially small. Therefore, even if we assume that these sequences are different for every input x, their sum is still less than the total number of random sequences. Formally, let's consider the probability over all sequences that the algorithm gives the right answer for all input. If this probability is greater than 0, then the claim is proved.

$$\mathbf{Pr}_r[\text{for every} \quad x, A(x,r) \quad \text{is correct}] = 1 - \mathbf{Pr}_r[\exists x, A(x,r) \quad \text{is wrong}]$$

the second probability is the union of 2^n possible events for each x. This is bounded by the sum of the probabilities.

$$\begin{aligned}
&\geq 1 - \sum_{x\in\{0,1\}^n} \mathbf{Pr}_r[A(x,r)\text{is wrong}] \\
&\geq 1 - 2^n \cdot 2^{-(n+1)} \\
&\geq \frac{1}{2}
\end{aligned}$$

□

So, we proved that at least half of the random sequences are correct for all possible input x. Therefore, it is straightforward to see that we can simulate the algorithm $A(\cdot,\cdot)$, where the first input has length n and the second $p(n)$, by a circuit of size polynomial in n.

All we have to do is find a random sequence which is always correct and build it inside the circuit. Hence, our circuit will take as input only the input x and simulate A with input x and r for this fixed r. Of course, this is only an existential proof, since we don't know how to find this sequence efficiently. □

In conclusion, let us briefly describe some other relations between complexity classes. Whether **BPP** $\subseteq$ **NP** or not is still an open question. What we know is that it's unlikely that **NP** is contained in **BPP**, since then by the previous result **NP** would have polynomial size circuits and hence by the result of Karp and Lipton the polynomial hierarchy would collapse.

3 BPP$\subseteq \Sigma_2$

This result was first shown by Sipser and Gacs. Lautemann gave a much simpler proof which we give below.

Lemma 8 *If L is in* **BPP** *then there is an algorithm A such that for every x,*

$$\mathbf{Pr}_r(A(x,r) = \textit{right answer}) \geq 1 - \tfrac{1}{3m},$$

where the number of random bits $|r| = m = |x|^{O(1)}$ and A runs in time $|x|^{O(1)}$.

PROOF: Let $\hat{A}$ be a **BPP** algorithm for L. Then for every x, $\mathbf{Pr}_r(\hat{A}(x,r) = \text{wrong answer}) \leq \frac{1}{3}$, and $\hat{A}$ uses $\hat{m}(n)$ random bits where $n = |x|$.

Do $k(n)$ repetitions of $\hat{A}$ and accept if and only if at least $\dfrac{k(n)}{2}$ executions of $\hat{A}$ accept. Call the new algorithm A. Then A uses $k(n)\hat{m}(n)$ random bits and $\mathbf{Pr}_r(A(x,r) = \text{wrong answer}) \leq 2^{-ck(n)}$. We can then find $k(n)$ with $k(n) = \Theta(\log \hat{m}(n))$ such that $\frac{1}{2^{ck(n)}} \leq \frac{1}{3k(n)m(n)}$. $\square$

Theorem 9 **BPP**$\subseteq \Sigma_2$.

PROOF: Let L be in **BPP** and A as in the claim. Then we want to show that

$$x \in L \Leftrightarrow \exists y_1, \ldots, y_m \in \{0,1\}^m \forall z \in \{0,1\}^m \bigvee_{i=1}^{m} A(x, y_i \oplus z) = 1$$

where m is the number of random bits used by A on input x.
Suppose $x \in L$. Then

$$\begin{aligned}
&\mathbf{Pr}_{y_1,\ldots,y_m}(\exists z A(x, y_1 \oplus z) = \cdots = A(x, y_m \oplus z) = 0) \\
&\leq \sum_{z \in \{0,1\}^m} \mathbf{Pr}_{y_1,\ldots,y_m}(A(x, y_1 \oplus z) = \cdots = A(x, y_m \oplus z) = 0) \\
&\leq 2^m \frac{1}{(3m)^m} \\
&< 1.
\end{aligned}$$

So

$$\begin{aligned}
\mathbf{Pr}_{y_1,\ldots,y_m}(\forall z \bigvee_i A(x, y_i \oplus z)) &= 1 - \mathbf{Pr}_{y_1,\ldots,y_m}(\exists z A(x, y_1 \oplus z) = \cdots = A(x, y_m \oplus z) = 0) \\
&> 0.
\end{aligned}$$

So $(y_1, \ldots, y_m)$ exists.
Conversely suppose $x \notin L$. Then

$$\begin{aligned}\mathbf{Pr}_z\left(\bigvee_i A(x, y_i \oplus z)\right) &\leq \sum_i \mathbf{Pr}_z\left(A(x, y_i \oplus z) = 1\right) \\ &\leq m \cdot \frac{1}{3m} \\ &= \frac{1}{3}.\end{aligned}$$

So

$$\begin{aligned}\mathbf{Pr}_z(A(x, y_1 \oplus z) = \cdots = A(x, y_m \oplus z) = 0) &= \mathbf{Pr}_z\left(\bigvee_i A(x, y_i \oplus z)\right) \\ &\geq \frac{2}{3} \\ &> 0.\end{aligned}$$

So there is a z such that $\bigvee_i A(x, y_i \oplus z) = 0$ for all $y_1, \ldots, y_m \in \{0,1\}^m$. □

4 The Valiant-Vazirani Reduction

In this section we show the following: suppose there is an algorithm for the satisfiability problem that always find a satisfying assignment for formulae that have exactly one satisfiable assignment (and behaves arbitrarily on other instances): then we can get an **RP** algorithm for the general satisfiability problem, and so **NP** = **RP**.

We prove the result by presenting a randomized reduction that given in input a CNF formula ϕ produces in output a polynomial number of formulae $\psi_0, \ldots, \psi_n$. If ϕ is satisfiable, then (with high probability) at least one of the ψ_i is satisfiable and has exactly one satisfying assignment; if ϕ is not satisfiable, then (with probability one) all ψ_i are unsatisfiable.

The idea for the reduction is the following. Suppose ϕ is a satisfiable formula with n variables that has about 2^k satisfying assignments, and let $h : \{0,1\}^n \to \{0,1\}^k$ be a hash function picked from a family of pairwise independent hash functions: then the average number of assignments x such that $\phi(x)$ is true and $h(x) = (0, \ldots, 0)$ is about one. Indeed, we can prove formally that with constant probability there is exactly one such assignment,[1] and that there is CNF formula ψ (easily constructed from ϕ and h) that is satisfied precisely by that assignment. By doing the above construction for values of k ranging from 0 to n, we obtain the desired reduction. Details follow.

Definition 6 *Let H be a family of functions of the form $h : \{0,1\}^n \to \{0,1\}^m$. We say that H is a family of pair-wise independent hash functions if for every two different inputs $x, y \in \{0,1\}^n$ and for every two possible outputs $a, b \in \{0,1\}^m$ we have*

$$\mathbf{Pr}_{h \in H}[h(x) = a \wedge h(y) = b] = \frac{1}{2^{2m}}$$

[1] For technical reasons, it will be easier to prove that this is the case when picking a hash function $h : \{0,1\}^n \to \{0,1\}^{k+2}$.

Another way to look at the definition is that for every $x \neq y$, when we pick h at random then the random variables $h(x)$ and $h(y)$ are independent and uniformly distributed. In particular, for every $x \neq y$ and for every a, b we have $\mathbf{Pr}_h[h(x) = a | h(y) = b] = \mathbf{Pr}_h[h(x) = a]$.

For m vectors $a_1, \ldots, a_m \in \{0,1\}^m$ and m bits $b_1, \ldots, b_m$, define $h_{a_1,\ldots,a_m,b_1,\ldots,b_m} : \{0,1\}^n \to \{0,1\}^m$ as $h_{\mathbf{a},b}(x) = (a_1 \cdot x + b_1, \ldots, a_m \cdot x + b_m)$, and let H_{AFF} be the family of functions defined this way. Then it is not hard to see that H_{AFF} is a family of pairwise independent hash functions.

Lemma 10 *Let $T \subseteq \{0,1\}^n$ be a set such that $2^k \leq |T| < 2^{k+1}$ and let H be a family of pairwise independent hash functions of the form $h : \{0,1\}^n \to \{0,1\}^{k+2}$. Then if we pick h at random from H, there is a constant probability that there is a unique element $x \in T$ such that $h(x) = \mathbf{0}$. Precisely,*

$$\mathbf{Pr}_{h \in H}[|\{x \in T : h(x) = \mathbf{0}\}| = 1] \geq \frac{1}{8}$$

PROOF: Let us fix an element $x \in T$. We want to compute the probability that x is the *unique* element of T mapped into $\mathbf{0}$ by h. Clearly,

$$\mathbf{Pr}_h[h(x) = \mathbf{0} \wedge \forall y \in T - \{x\}.h(y) \neq \mathbf{0}] = \mathbf{Pr}_h[h(x) = \mathbf{0}] \cdot \mathbf{Pr}_h[\forall y \in T - \{x\}.h(y) \neq \mathbf{0} | h(x) = \mathbf{0}]$$

and we know that

$$\mathbf{Pr}_h[h(x) = \mathbf{0}] = \frac{1}{2^{k+2}}$$

The difficult part is to estimate the other probability. First, we write

$$\mathbf{Pr}_h[\forall y \in T - \{x\}.h(y) \neq \mathbf{0} | h(x) = \mathbf{0}] = 1 - \mathbf{Pr}_h[\exists y \in T - \{x\}.h(y) = \mathbf{0} | h(x) = \mathbf{0}]$$

And then observe that

$$\begin{aligned}
& \mathbf{Pr}_h[\exists y \in T - \{x\}.h(y) = \mathbf{0} | h(x) = \mathbf{0}] \\
\leq & \sum_{y \in |T| - \{x\}} \mathbf{Pr}_h[h(y) = \mathbf{0} | h(x) = \mathbf{0}] \\
= & \sum_{y \in |T| - \{x\}} \mathbf{Pr}_h[h(y) = \mathbf{0}] \\
= & \frac{|T| - 1}{2^{k+2}} \\
\leq & \frac{1}{2}
\end{aligned}$$

Notice how we used the fact that the value of $h(y)$ is independent of the value of $h(x)$ when $x \neq y$.

Putting everything together, we have

$$\mathbf{Pr}_h[\forall y \in T - \{x\}.h(y) \neq \mathbf{0} | h(x) = \mathbf{0}] \geq \frac{1}{2}$$

and so

$$\mathbf{Pr}_h[h(x) = \mathbf{0} \wedge \forall y \in T - \{x\}.h(y) \neq \mathbf{0}] \geq \frac{1}{2^{k+3}}$$

To conclude the argument, we observe that the probability that there is a unique element of T mapped into $\mathbf{0}$ is given by the sum over $x \in T$ of the probability that x is the unique element mapped into $\mathbf{0}$ (all this events are disjoint, so the probability of their union is the sum of the probabilities). The probability of a unique element mapped into $\mathbf{0}$ is then at least $|T|/2^{k+3} > 1/8$. □

Lemma 11 *There is a probabilistic polynomial time algorithm that on input a CNF formula ϕ and an integer k outputs a formula ψ such that*

- *If ϕ is unsatisfiable then ψ is unsatisfiable.*
- *If ϕ has at least 2^k and less than 2^{k+1} satifying assignments, then there is a probability at least $1/8$ then the formula ψ has exactly one satisfying assignment.*

PROOF: Say that ϕ is a formula over n variables. The algorithm picks at random vectors $a_1, \ldots, a_{k+2} \in \{0,1\}^n$ and bits $b_1, \ldots, b_{k+2}$ and produces a formula ψ that is equivalent to the expression $\phi(x) \wedge (a_1 \cdot x + b_1 = 0) \wedge \ldots \wedge (a_{k+2} \cdot x + b_{k+2} = 0)$. Indeed, there is no compact CNF expression to compute $a \cdot x$ if a has a lot of ones, but we can proceed as follows: for each i we add auxiliary variables $y_1^i, \ldots, y_n^i$ and then write a CNF condition equivalent to $(y_1^i = x_1 \wedge a_i[1]) \wedge \cdots \wedge (y_n^i = y_{n-1}^i \oplus (x_n \wedge a_i[n] \oplus b_i)))$. Then ψ is the AND of the clauses in ϕ plus all the above expressions for $i = 1, 2, \ldots, k+2$.

By construction, the number of satisfying assignments of ψ is equal to the number of satisfying assignments x of ϕ such that $h_{a_1,\ldots,a_{k+2},b_1,\ldots,b_{k+2}}(x) = \mathbf{0}$. If ϕ is unsatisfiable, then, for every possible choice of the a_i, ψ is also unsatisfiable.

If ϕ has between 2^k and 2^{k+1} assignments, then Lemma 10 implies that with probability at least $1/8$ there is exactly one satisfying assignment for ψ. □

Theorem 12 (Valiant-Vazirani) *Suppose there is a polynomial time algorithm that on input a CNF formula having exactly one satisfying assignment finds that assignment. (We make no assumption on the behaviour of the algorithm on other inputs.) Then* $\mathbf{NP} = \mathbf{RP}$.

PROOF: It is enough to show that, under the assumption of the Theorem, 3SAT has an **RP** algorithm.

On input a formula ϕ, we construct formulae $\psi_0, \ldots, \psi_n$ by using the algorithm of Lemma 11 with parameters $k = 0, \ldots, n$. We submit all formulae $\psi_0, \ldots, \psi_n$ to the algorithm in the assumption of the Theorem, and accept if the algorithm can find a satisfying assignment for at least one of the formulae. If ϕ is unsatisfiable, then all the formulae are always unsatisfiable, and so the algorithm has a probability zero of accepting. If ϕ is satisfiable, then for some k it has between 2^k and 2^{k+1} satisfying assignments, and there is a probability at least $1/8$ that ψ_k has exactly one satisfying assignment and that the algorithm accepts. If we repeat the above procedure t times, and accept if at least one iteration accepts, then if ϕ is unsatisfiable we still have probability zero of accepting, otherwise we have probability at least $1 - (7/8)^t$ of accepting, which is more than $1/2$ already for $t = 6$. □

5 References

Probabilistic complexity classes were defined in [Gil77]. Adleman's proof that $\mathbf{BPP} \subseteq \mathbf{SIZE}(n^{O(1)})$ appears in [Adl78]. Sipser's proof that $\mathbf{BPP} \subseteq \Sigma_2$ appears in [Sip83], and Lautemann's proof is in [Lau83]. The Valiant-Vazirani result is from [VV86].

References

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, pages 75–83, 1978. 10

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6:675–695, 1977. 10

[Lau83] C. Lautemann. BPP and the Polynomial Hierarchy. *Information Processing Letters*, 17:215–217, 1983. 10

[Sip83] M. Sipser. A complexity theoretic appraoch to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 330–335, 1983. 10

[VV86] L.G. Valiant and V.V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986. 10

Exercises

1. Prove that **ZPP**=**RP**∩**coRP**
2. Show that if **NP** ⊆ **BPP** then **NP** = **RP**.
3. Prove that **SPACE**($O(n^{\log n})$) ⊈ **BPP**.
4. Change the assumption of Theorem 12 to having a *probabilistic* polynomial time algorithm that on input a formula with exactly one satisfying assignment finds that assignment with probability at least 1/2. Prove that it still follows that **NP** = **RP**.