Notes for Lecture 4

In this lecture we prove that $\mathbf{P} \neq \mathbf{NP}$ with probability 1 relative to a random oracle, we talk about randomized algorithms, and we show that Boolean circuits can simulate randomized algorithms.

1 Relativizations

With a few significant exceptions, all known results about complexity classes "relativize," meaning that the results remain true if one replaces the complexity classes involved in the result with the respective classes relative to an oracle A, for every choose of A. Certain complexity theoretic statements, however, cannot relativize, because they are true relative to certain oracles and false relative to certain other oracles; it then follows that such statements cannot be proved or disproved using relativizing techniques. This is an important bottleneck to progress in complexity theory.

We now define the above notions. An oracle Turing machine M operating with an oracle A is a machine, that has an extra tape, called the oracle tape, and a special state, called the query state. When the machine enter the query state, at the next step the state encodes whether or not the content of the oracle tape is an element of A or not. Informally, an oracle Turing machine M operating with an oracle A formalizes an algorithm that is allowed to use, at unit cost per execution, a subroutine the checks membership in A.

If A is a language, we denote by $\mathbf{P}^A$ the class of languages that can be solved in polynomial time by oracle Turing machines operating with the oracle A. (This can also be thought as the class of languages that can be reduced to A by polynomial time reductions.) We can similarly define the relativized analogs of every other complexity class whose definition involves time-limited deterministic machine, such as $\mathbf{EXP}$, $\mathbf{DTIME}(t(n))$ and so on.

We "relativize" the definition of $\mathbf{NP}$ in the following way. We say that a language L is in $\mathbf{NP}^A$ if there is a relation R and a polynomial $q(\cdot)$ such that membership in R can be checked in polynomial time *by an oracle machine* with oracle A, and such that

$$x \in L \Leftrightarrow \exists y. |y| \leq p(|x|) \wedge (x, y) \in R \ .$$

Notation such as $\mathbf{P}^A$ or $\mathbf{NP}^A$ obscure a subtle point that can possibly cause confusion. "Relativization" is not an operation that is applied to a complexity class, such as $\mathbf{NP}$, and to an oracle A to produce the relativized class $\mathbf{NP}^A$. Instead what we "relativize" is a specific *machine-based definition* of $\mathbf{NP}$, and we obtain the class $\mathbf{NP}^A$ by replacing the Turing machines occurring in the machine-based definition with oracle Turing machines having oracle A.[1]

[1] Here is why this distinction is important: if relativization was an operation on complexity classes, then we would have that $\mathbf{P} = \mathbf{NP}$ would imply $\mathbf{P}^A = \mathbf{NP}^A$ for every oracle A, and exhibiting an oracle B such that $\mathbf{P}^B \neq \mathbf{NP}^B$ would imply $\mathbf{P} \neq \mathbf{NP}$. Instead, in the next section we will show that such an oracle B exists, and this has no implication on the status of the $\mathbf{P}$ versus $\mathbf{NP}$ problem.

If A is a **PSPACE**-complete problem, then it is easy to see that $\mathbf{P}^A = \mathbf{NP}^A = \mathbf{PSPACE}$. This means that it is not possible to prove $\mathbf{P} \neq \mathbf{NP}$ using relativizing techniques. We will now show that there is an oracle B such that $\mathbf{P}^B \neq \mathbf{NP}^B$. In fact we will show more: that if B is a randomly chosen oracle, the probability that $\mathbf{P}^B = \mathbf{NP}^B$ is zero.

2 Separating P and NP with a Random Oracle

We prove the following result.

Theorem 1

$$\mathop{\mathbb{P}}_{R} [\mathbf{P}^R \neq \mathbf{NP}^R] = 1$$

where the probability is taken over a random oracle $R \subseteq \{0,1\}^*$.

(Note that the probability space of all possible languages R is a continuous space, and so if something happens with probability 1 it does not mean that it happens for all possible R.)

Let us discuss first the intuition of the proof. Imagine first that, instead of having access to a random oracle selected according to the uniform distribution, we have access to an oracle such that, for every n, with probability $1/2$ there is no string of length n in R, and with probability $1/2$ there is exactly one (randomly chosen) string of length n in R, and that, given n, and given access to the oracle, we want to find out which case holds, in time polynomial in n. This is an "NP-type" problem, because a witness of the fact that there is a string of length n in the oracle is a string of length n in the oracle (the validity of the witness can be verified with one oracle query). It seems impossible, however, to solve the problem deterministically in time polynomial in n: whether there is a string of length n in the oracle or not, with high probability a polynomial time algorithm will only get NO answers from the oracle, for all its queries of length n. But the output of the algorithm depends only on the answers from the oracle, so, conditioned on getting all "NO" answers, the algorithm will give the same answer whether a string of length n is in the oracle or not, and so it will be incorrect with probability approximately $1/2$.

There a few adjustments that we need to make to the above argument: we need to define a computational problem where the input is a string of length n rather than the number n, we need to make the reasoning work relative to a uniformly random oracle, and we need to be able to first pick a random oracle, and then rule out all possible polynomial time algorithms (in the informal argument, we are fixing the polynomial time algorithm first and then we pick the random oracle); finally, we want the probability of the existence of a polynomial time algorithm for our hard problem to be zero.

For every oracle R, we define the language L_R in the following way: a bit string x of length n belongs to L_R if and only if there exists a bit string y also of length n such that for all bits strings z of length $\log n$ we have that $xyz \in R$, where xyz is the concatenation (of length $2n + \log n$) of the three strings x, y, and z.

The point is that, for a fixed x of length n, if we pick a random oracle R, then in order to have $x \in L_R$ we need at least one of 2^n possible conditions (corresponding to the choice of y) to be true, and each possibility has probability 2^{-n} of being true (because it

corresponds to the event that a certain set of n strings all belong to the oracle). So x has probability $1-(1-1/2^n)^{2^n} \approx 1-1/e$ of being in L_R and probability $\approx 1/2$ of not being in L_R. Furthermore, we have:

Fact 2 *For every oracle R, the language L_R is in $\mathbf{NP}^R$*

PROOF: A witness that $x \in L_R$ is a string y such that $xyz \in L_R$ for all z, and this condition can be tested with n oracle queries. □

Note that if $x \in L_R$ then we expect a small number of witnesses of this fact, so we are in a setup that is fairly similar to the one described above. (With constant probability $x \in L_R$, with constant probability $x \notin L_R$, and the number of witnesses is very small.)

We begin by proving the following weak fact.

Lemma 3 *Let M be a polynomial time oracle machine. Then, for all sufficiently large n and all x of length n we have*

$$\mathop{\mathbb{P}}_{R}[M^R(x) = L_R(x)] \leq \frac{2}{3}$$

PROOF: Let M' be the machine that behaves like M except that, if it asks the query xyz where $|x| = |y| = n$ and $|z| = \log n$ then it also asks the queries xyz' for every $z' \in \{0,1\}^{\log n}$. Note that M' is still a polynomial time machine, and let $p(n)$ be a polynomial upper bound to the queries asked by M' given an input of length n. Without loss of generality, we can assume that M' makes exactly $p(n)$ oracle queries on inputs of length n (otherwise make additional oracle queries and ignore the answers). A "transcript" c of a computation of $M'^R(x)$ is a description of all the answers to the $p(n)$ oracle queries asked by M'. Note that, even if M' asks its questions adaptively, and so there are up to $2^{p(n)}$ questions that can possibly be asked, and each one has two possible answers, the total number of valid transcripts is only $2^{p(n)}$, and each one has probability $1/2^{p(n)}$.

An oracle R is *consistent* with a transcript c if all the oracle queries and answers reported in c agree with R. Note that if $M'^R(x)$ has transcript c, and R' is consistent with c, then the output of $M'^R(x)$ is the same as the output of $M'^{R'}(x)$.

Given a transcript c, we can consider the conditional distribution of a random oracle R given c; this is just the distribution of an oracle that agrees with the answers recorded in c for the $p(n)$ queries of c, and is random on all other inputs.

We say that a transcript c is *inconclusive* for x if it contains no query xyz where y is a valid witness that $x \in L_R$.

We make the following two claims

$$\mathop{\mathbb{P}}_{R}[M'^R(x) \text{ has an inconclusive transcript }] \geq 1 - \frac{n^{O(1)}}{2^n} \tag{1}$$

If c is an inconclusive transcript of M' on input x,

$$\mathop{\mathbb{P}}_{R}[x \in L_R \mid R \text{ consistent with } c] = 1 - \frac{1}{e} + o(1) \tag{2}$$

Regarding the first claim, imagine the process of running the computation of $M'^R(x)$ and of "picking R randomly along the way" by just making random decisions for which of the queries of M' are in R and which are not, and delaying every other decision. Every time M' makes a block of queries of the form xyz with $z \in \{0,1\}^{\log n}$, there is probability $1/2^n$ that y is a witness for x. M' makes $p(n)/n)$ such query blocks, and so, by a union bound, the probability that it queries a witness of x (thus creating a non-inconclusive witness) is at most $p(n)/(n \cdot 2^n) = n^{O(1)}/2^n$.

Regarding the second claim, let c be an inconclusive transcript, and consider the selection of a random R consistent with c. Each of the $2^n - n^{O(1)}$ strings y for which no query xyz is in the transcript has probability $1/2^n$ of becoming a witness that $x \in L_R$, and the strings y for which the queries xyz are in the transcript are definitely not going to be witnesses. So the probability that $x \notin L_R$ is precisely

$$\left(1 - \frac{1}{2^n}\right)^{2^n - n^{O(1)}} = \frac{1}{e} - o(1)$$

Having proved the two claims, the statement of the lemma follows easily: let us pick a random oracle R by first picking a random transcript of $M'(x)$ and then a random R consistent with c. When we pick c, we have $1 - o(1)$ of picking an inconclusive one; once we have picked such a transcript, the output of $M'^R(x)$ depends only on c, and it is the same for all R consistent with c; but for a $1/e - o(1)$ fraction of R the correct answer is "reject" and for a $1 - 1/e + o(1)$ fraction the correct answer is "accept," and so $M'^R(x)$ is going to be wrong with probability at least $1/e - o(1)$, which is more than $1/3$ for sufficiently large n. □

We now make the following stronger claim.

Lemma 4 *Let M be a polynomial time oracle machine. Then, for all sufficiently large n and all $x_1, \ldots, x_n$, each of length n, we have*

$$\mathop{\mathbb{P}}_R \left[M^R(x_1) = L_R(x_1) \wedge \cdots \wedge M^R(x_n) = L(x_n)\right] \leq \left(\frac{2}{3}\right)^n$$

PROOF: Let M' be an oracle machine such that M'^R simulates $M^R(x_1)$, $M^R(x_2)$, ..., $M^R(x_n)$ and then returns the outputs of the n simulations; furthermore, if M' makes a query of the form $x_i yz$, then it also makes all the queries of the form $x_i yz'$ for all $z' \in \{0,1\}^{\log n}$. Finally, assume that M' always makes the same number of queries for every oracle, and let this number of queries be $p(n)$. We refer to a transcript of M' as a record of all the oracle queries and answers. As before, there are $2^{p(n)}$ valid transcripts, and they are all equally likely.

We say that a transcript is inconclusive if it contains no query $x_i yz$ where y is a valid witness for x_i. (That is, the transcript is inconclusive, according to the definition in the previous lemma, for all the x_i.)

As before, if we pick a random transcript, then it is inconclusive with probability at least $1 - n^{O(1)}/2^n$. If a transcript c is inconclusive, and we sample a random R consistent with c, then each x_i has, independently, probability $1/e - o(1)$ of not being in L_R and probability $1 - 1/e + o(1)$ of being in L_R. Overall, it follows that

$$\mathop{\mathbb{P}}_{R}\left[M^R(x_1) = L_R(x_1) \wedge \cdots \wedge M^R(x_n) = L(x_n)\right] \leq \left(1 - \frac{1}{e} + o(1)\right)^n + \frac{n^{O(1)}}{2^n} \leq \left(\frac{2}{3}\right)^n$$

for large enough n. $\square$

Now, let us fix a polynomial time machine M; then for each sufficiently large n we have

$$\mathop{\mathbb{P}}_{R}[M^R \text{ correctly decides } L_R] \leq \left(\frac{2}{3}\right)^n$$

and since this is true for every sufficiently large n we have to conclude that

$$\mathop{\mathbb{P}}_{R}[M^R \text{ correctly decides } L_R] = 0$$

Finally, let $\mathcal{M}$ be the (countable) set of all polynomial time oracle machines, then

$$\begin{aligned}\mathop{\mathbb{P}}_{R}[L_R \in \mathbf{P}^R] &= \mathop{\mathbb{P}}_{R}[\exists M \in \mathcal{M}.M \text{ decides } L_R] \\ &\leq \sum_{M \in \mathcal{M}} \mathbb{P}[M \text{ decides } L_R] = 0\end{aligned}$$

where we use countable additivity of the uniform distribution over random oracles.

3 Randomized Algorithms

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 5 *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 $\mathbb{P}[A(x,r) = \text{correct answer for } x] \geq \frac{2}{3}$. 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 6 *A decision problem L is in* **BPP** *if there is a polynomial time algorithm A and a polynomial $p()$ such that :*

$$\forall x \in L \qquad \mathop{\mathbb{P}}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] \geq 2/3$$

$$\forall x \notin L \qquad \mathop{\mathbb{P}}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] \leq 1/3$$

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

$$\forall x \in L \quad : \underset{r\in\{0,1\}^{p(|x|)}}{\mathbb{P}}[A(x,r)=1]=1$$

$$\forall x \notin L : \underset{r\in\{0,1\}^{p(|x|)}}{\mathbb{P}}[A(x,r)=1]=0$$

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

Definition 8 *A decision problem L is in* **RP** *if there is a polynomial time algorithm A and a polynomial $p()$ such that:*

$$\forall x \in L \qquad \underset{r\in\{0,1\}^{p(|x|)}}{\mathbb{P}}[A(x,r)=1]\geq 1/2$$

$$\forall x \notin L \qquad \underset{r\in\{0,1\}^{p(|x|)}}{\mathbb{P}}[A(x,r)=1]\leq 0$$

Definition 9 *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 :*

$$\forall x \qquad \underset{r\in\{0,1\}^{p(|x|)}}{\mathbb{P}}[A(x,r)=?]\leq 1/2$$

$$\forall x, \forall r \quad \textit{such that} \quad A(x,r)\neq ?.(\ A(x,r)=1 \quad \Leftrightarrow \quad x\in L\ )$$

4 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 10 **RP**$\subseteq$**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 11 **ZPP**$\subseteq$**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 12 *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 13 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:

- 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 $A(x, r_i) = 0$.

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

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

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. □

5 Adleman's Theorem

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

IIf we start from a randomized algorithm that provides the correct answer only with probability slightly higher than half, then repeating the algorithm many times with independent randomness will make the right answer appear the majority of the times with very high probability.

More formally, we have the following theorem.

Theorem 14 (Chernoff Bound) *Suppose $X_1, \ldots, X_k$ are independent random variables with values in $\{0, 1\}$ and for every i, $\mathbb{P}[X_i = 1] = p_i$. Then, for any $\epsilon > 0$:*

$$\mathbb{P}\left[\sum_{i=1}^{k} X_i > \sum_{i=1}^{k} p_i + k\epsilon\right] < e^{-2\epsilon^2 k}$$

$$\mathbb{P}\left[\sum_{i=1}^{k} X_i < \sum_{i=1}^{k} p_i - k\epsilon\right] < e^{-2\epsilon^2 k}$$

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 with 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 $p = \mathbb{P}_r[A(x,r) = 1]$ over all possible random sequences. We also define the independent 0/1 random variables $X_1, \ldots, X_k$ such that $X_i = 1$ if and only if $A(x, r_i)$ outputs the correct answer.

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

We now apply the Chernoff bounds to bound this probability.

$$\begin{aligned}
&\mathbb{P}[A^{(k)}\text{outputs the wrong answer on } x] \\
&= \mathbb{P}[\sum_i X_i < \tfrac{k}{2}] \\
&\leq \mathbb{P}[\sum_i X_i - kp \leq -\tfrac{k}{6}] \\
&\leq e^{-k/18} \\
&= 2^{-\Omega(k)}
\end{aligned}$$

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 could equivalently have a bound of $1/2 - 1/q(|x|)$ or $2^{-q(|x|)}$, for a fixed polynomial q.

Would it be equivalent to have a bound of $1/2 - 2^{-q(|x|)}$?

Definition 15 PP *is the set of problems that can be solved by a nondeterministic Turing machine in polynomial time where the acceptance condition is that a majority (more than half) of computation paths accept.*

Although superficially similar to **BPP**, **PP** is a very powerful class; $\mathbf{P}^{\mathbf{PP}}$ (polynomial time computations with an oracle for **PP**) includes all of **NP**, quantum polynomial time **BQP**, and the entire polynomial hierarchy $\Sigma_1 \subseteq \Sigma_2 \subseteq \ldots$ which we will define later.

Now, 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 16 (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

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

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 circuit C_n, that solves L on inputs of length n.

Claim 17 *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 of length n, 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.

$$\mathop{\mathbb{P}}_{r}[\text{for every} \quad x, A(x,r) \quad \text{is correct}] = 1 - \mathop{\mathbb{P}}_{r}[\exists x, A(x,r) \quad \text{is wrong}]$$

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

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

□

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. □