Notes for Lecture 3

In this lecture we introduce the computational model of boolean circuits and prove that polynomial size circuits can simulate all polynomial time computations, we talk about randomized algorithms, and we show that Boolean circuits can simulate randomized algorithms.

1 Circuits

A circuit C has n inputs, m outputs, and is constructed with AND gates, OR gates and NOT gates. Each gate has in-degree 2 except the NOT gate which has in-degree 1. The out-degree can be any number. A circuit must have no cycle. See Figure 1.

A circuit C with n inputs and m outputs computes a function $f_C : \{0,1\}^n \to \{0,1\}^m$. See Figure 2 for an example.

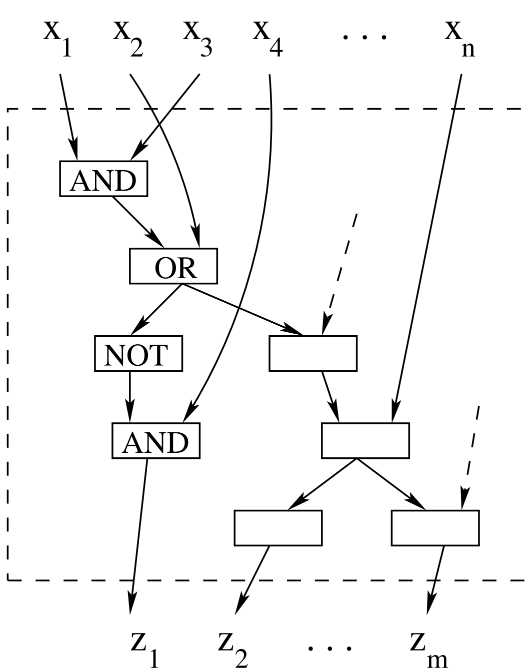


Figure 1: A Boolean circuit.

Define $\mathbf{SIZE}(C) = \#$ of AND and OR gates of C. By convention, we do *not* count the NOT gates.

To be compatible with other complexity classes, we need to extend the model to arbitrary input sizes:

Definition 1 *A language L is solved by a family of circuits $\{C_1, C_2, \ldots, C_n, \ldots\}$ if for every $n \geq 1$ and for every x s.t. $|x| = n$,*

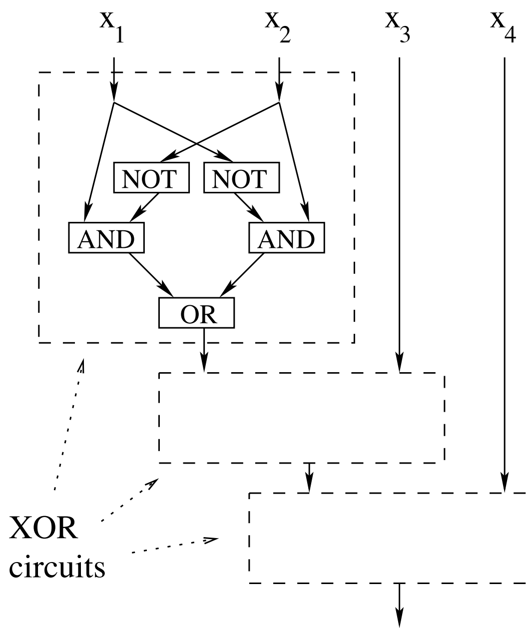


Figure 2: A circuit computing the boolean function $f_C(x_1x_2x_3x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

$$x \in L \iff f_{C_n}(x) = 1.$$

Definition 2 *Say $L \in \mathbf{SIZE}(s(n))$ if L is decided by a family $\{C_1, C_2, \ldots, C_n, \ldots\}$ of circuits, where C_i has at most $s(i)$ gates.*

2 Relation to other complexity classes

Unlike other complexity measures, like time and space, for which there are languages of arbitrarily high complexity, the size complexity of a problem is always at most exponential.

Theorem 3 *For every language L, $L \in \mathbf{SIZE}(O(2^n))$.*

Proof: We need to show that for every 1-output function $f : \{0,1\}^n \to \{0,1\}$, f has circuit size $O(2^n)$.

Use the identity $f(x_1x_2\ldots x_n) = (x_1 \wedge f(1x_2\ldots x_n)) \vee (\overline{x}_1 \wedge f(0x_2\ldots x_n))$ to recursively construct a circuit for f, as shown in Figure 3.

The recurrence relation for the size of the circuit is: $s(n) = 3 + 2s(n-1)$ with base case $s(1) = 1$, which solves to $s(n) = 2 \cdot 2^n - 3 = O(2^n)$. $\square$

The exponential bound is nearly tight.

Theorem 4 *There are languages L such that $L \notin \mathbf{SIZE}(o(2^n/n))$. In particular, for every $n \geq 11$, there exists $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^n/4n$.*

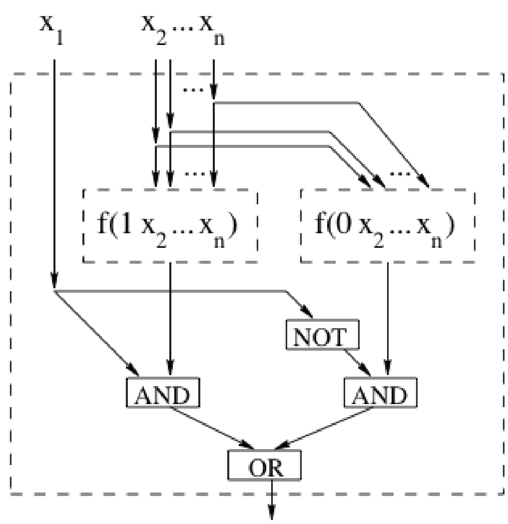


Figure 3: A circuit computing any function $f(x_1 x_2 \dots x_n)$ of n variables assuming circuits for two functions of $n-1$ variables.

PROOF: This is a counting argument. There are 2^{2^n} functions $f : \{0,1\}^n \to \{0,1\}$, and we claim that the number of circuits of size s is at most $2^{O(s \log s)}$, assuming $s \geq n$. To bound the number of circuits of size s we create a compact binary encoding of such circuits. Identify gates with numbers $1, \dots, s$. For each gate, specify where the two inputs are coming from, whether they are complemented, and the type of gate. The total number of bits required to represent the circuit is

$$s \times (2 \log(n+s) + 3) \leq s \cdot (2 \log 2s + 3) = s \cdot (2 \log 2s + 5).$$

So the number of circuits of size s is at most $2^{2s \log s + 5s}$, and this is not sufficient to compute all possible functions if

$$2^{2s \log s + 5s} < 2^{2^n}.$$

This is satisfied if $s \leq \frac{2^n}{4n}$ and $n \geq 11$. $\Box$

The following result shows that efficient computations can be simulated by small circuits.

Theorem 5 *If* $L \in \mathbf{DTIME}(t(n))$, *then* $L \in \mathbf{SIZE}(O(t^2(n)))$.

PROOF: Let L be a decision problem solved by a machine M in time $t(n)$. Fix n and x s.t. $|x| = n$, and consider the $t(n) \times t(n)$ tableau of the computation of $M(x)$. See Figure 4.

Assume that each entry (a, q) of the tableau is encoded using k bits. By Proposition 3, the transition function $\{0,1\}^{3k} \to \{0,1\}^k$ used by the machine can be implemented by a "next state circuit" of size $k \cdot O(2^{3k})$, which is exponential in k but constant in n. This building block can be used to create a circuit of size $O(t^2(n))$ that computes the complete tableau, thus also computes the answer to the decision problem.This is shown in Figure 5.

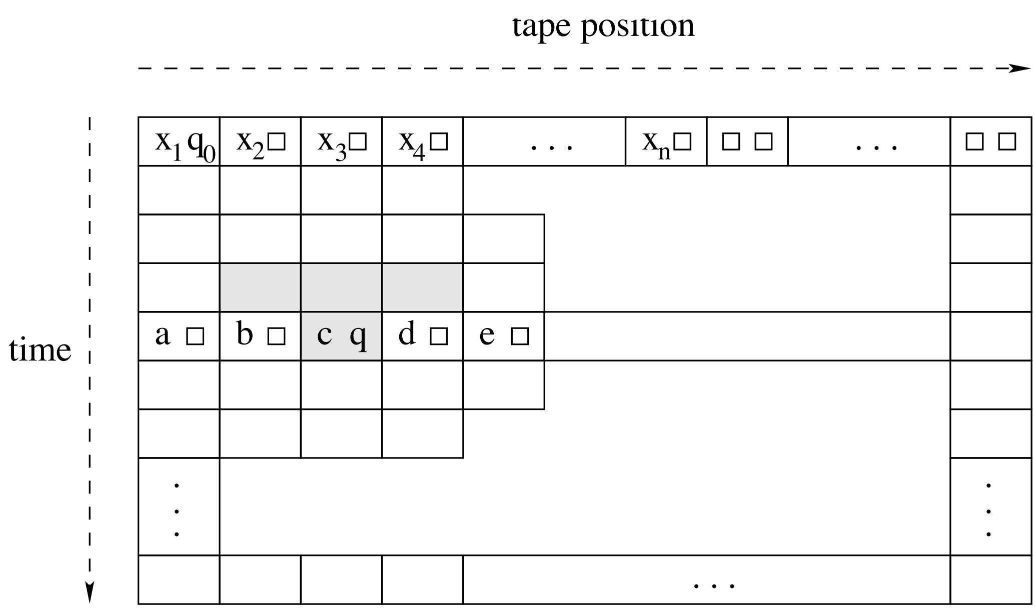


Figure 4: $t(n) \times t(n)$ tableau of computation. The left entry of each cell is the tape symbol at that position and time. The right entry is the machine state or a blank symbol, depending on the position of the machine head.

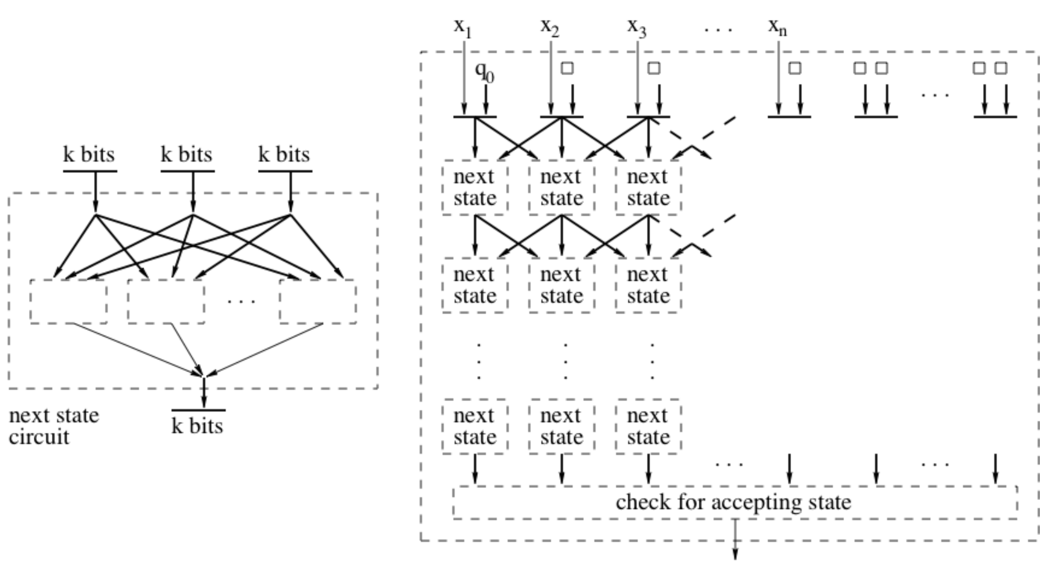


Figure 5: Circuit to simulate a Turing machine computation by constructing the tableau.

□

Corollary 6 $\mathbf{P} \subseteq \mathbf{SIZE}(n^{O(1)})$.

On the other hand, it's easy to show that $\mathbf{P} \neq \mathbf{SIZE}(n^{O(1)})$, and, in fact, one can define languages in $\mathbf{SIZE}(O(1))$ that are undecidable.

An equivalent characterization of languages decidable by polynomial size circuits can be given using the notion of *advice*.

Definition 7 *A language L can be decided in time $t(n)$ and advice $a(n)$ if there is an algorithm $A(\cdot,\cdot)$ runnint in time $\leq t(n)$ on inputs of length n, such that for every input length n there exists an "advice" string s_n of length $\leq a(n)$ such that for every x of length n*

$$x \in L \Leftrightarrow A(x, s_n) \text{ accepts} \quad .$$

We denote by $\mathbf{P}/poly$ the class of languages that can be decided in polynomial time using advice of polynomial length.

Theorem 8 $\mathbf{P}/poly = \mathbf{SIZE}(n^{O(1)})$.

Proof: Suppose that $L \in \mathbf{SIZE}(n^{O(1)})$ and consider the circuit evaluation algorithm A that on input a string x and a circuit C outputs $C(x)$. Clearly A is a polynomial time algorithm, and it witnesses $L \in \mathbf{P}/poly$, by using a minimal-size circuit for $L \cap \{0,1\}^n$ as the advice string for inputs of length n.

Suppose that $L \in \mathbf{P}/poly$, and let A be the advice algorithm. Then, for every input length n, we can construct a circuit C of size $n^{O(1)}$ such that, for the appropriate advice string s_n, we have $C(x, s_n) = 1$ iff $x \in L$. Hard-wire the string s_n into the circuit. □

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 9 *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 10 *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 11 *A decision problem L is in* **P** *if there is a polynomial time algorithm A and a polynomial $p()$ such that :*

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

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

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

Definition 12 *A decision problem L is in* **RP** *if there is a polynomial time algorithm A and a polynomial $p()$ such that:*

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

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

Definition 13 *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 :*

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

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

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 14 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 15 **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. $\Box$

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 16 *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. $\Box$

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

Theorem 17 **RP**$\subseteq$**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 $A(x, r_i) = 0$.

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

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

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. $\Box$

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 18 (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.

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

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 19 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 20 (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

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

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

Claim 21 *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.

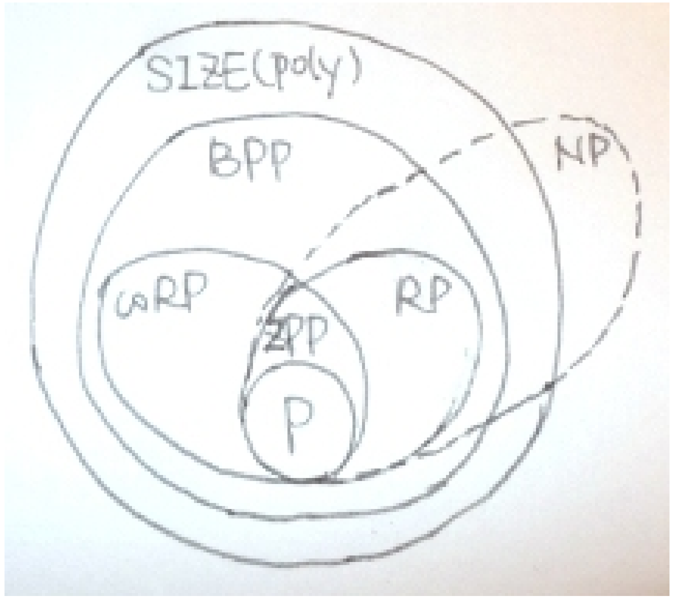
$$\begin{aligned}
&\geq 1 - \sum_{x\in\{0,1\}^n} \mathop{\mathbb{P}}_{r}[A(x,r)\text{is wrong}] \\
&\geq 1 - 2^n \cdot 2^{-(n+1)} \\
&\geq \frac{1}{2}
\end{aligned}$$

□

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. □

In general, the hierarchy of complexity classes looks like the following picture, if we visualize all classes that are not known to be equal as distinct.



It is, however, generally conjectured that $\mathbf{P} = \mathbf{BPP}$, in which case the complexity map greatly simplifies: