Randomized computation

IIC3242

What is this about?

One computational resource we still did not consider is **randomness**

After all, in programming languages you have a random number generator

So can we gain something extra with this?

An easy example: polynomials

We will consider polynomials over $\mathbb{Q}$

A polynomial is an expression of the form:

$$p(x) = \sum_{i=1}^{k} \prod_{j=1}^{\ell_i} (x + a_{i,j})$$

with each $a_{i,j} \in \mathbb{Q}$

The canonical form of $p(x)$ is an expression of the form:

$$p(x) = \sum_{i=0}^{\ell} c_i x^i$$

where each $c_i \in \mathbb{Q}$ and $\ell = \max\{\ell_1, \ldots, \ell_k\}$

The degree of $p(x)$ is ℓ.

An easy example: polynomials

When evaluating polynomials over the domain $\mathbb{Q}$

- The most expensive operation is multiplication, so we will measure the complexity in terms of evaluation

An easy example: polynomials

When evaluating polynomials over the domain $\mathbb{Q}$

- The most expensive operation is multiplication, so we will measure the complexity in terms of evaluation

Exercise

1. Give an algorithm that, given $p(x)$ and $a \in \mathbb{Q}$ as input calculates $p(a)$.
2. Give an algorithm that, given $p(x)$ in canonical form and $a \in \mathbb{Q}$ as input, calculates $p(a)$.

The two algorithms should perform $O(\ell)$ multiplications, where ℓ is the degree of $p(x)$.

Equivalence of polynomials

Given two polynomials $p(x)$ and $q(x)$, we want to verify if they are identical.

- For each $a \in \mathbb{Q}$, it holds that $p(a) = q(a)$

Equivalence of polynomials

Given two polynomials $p(x)$ and $q(x)$, we want to verify if they are identical.

- For each $a \in \mathbb{Q}$, it holds that $p(a) = q(a)$

We define the following language:

EQUIV-POL $= \{(p(x), q(x)) \mid p(x)$ and $q(x)$ are identical polynomials$\}$

Equivalence of polynomials

Given two polynomials $p(x)$ and $q(x)$, we want to verify if they are identical.

- For each $a \in \mathbb{Q}$, it holds that $p(a) = q(a)$

We define the following language:

$$\text{EQUIV-POL} = \{(p(x), q(x)) \mid p(x) \text{ and } q(x) \text{ are identical polynomials}\}$$

How can we solve EQUIV-POL?

An algorithm for EQUIV-POL

Input: $p(x)$ and $q(x)$

1. Find the degrees k and ℓ of $p(x)$ and $q(x)$, respectively
2. If $k = \ell$, then goto step 3, otherwise return **no**
3. Convert $p(x)$ and $q(x)$ into their canonical form:

$$p(x) = \sum_{i=0}^{k} c_i x^i$$

$$q(x) = \sum_{i=0}^{k} d_i x^i$$

4. Check if $c_i = d_i$ for all $i \in \{0, \ldots, k\}$. If this is true return **yes**, otherwise return **no**

An algorithm for EQUIV-POL

Exercise

1. Show that the previous algorithm is correct.
2. Show that the algorithm uses $O(k^2)$ multiplications.

An algorithm for EQUIV-POL

Exercise

1. Show that the previous algorithm is correct.
2. Show that the algorithm uses $O(k^2)$ multiplications.

Can we solve the problem using fewer multiplications?

Enter randomness
(a probabilistic algorithm for EQUIV-POL)

Input: $p(x)$ and $q(x)$

1. Determine the degrees k and ℓ of $p(x)$ and $q(x)$, respectively
2. If $k = \ell$, then goto step 3, otherwise return **no**
3. Randomly (and uniformly) select some a from the set $\{1, ..., 100k\}$
4. Check if $p(a) = q(a)$. If this is true, return **yes**, otherwise return **no**

Is this faster?

The probabilistic algorithm will do only $O(k)$ multiplications

- As it only needs to evaluate two polynomials of degree k

Is this faster?

The probabilistic algorithm will do only $O(k)$ multiplications

- As it only needs to evaluate two polynomials of degree k

But the algorithms can give a wrong answer as well!

- What is the probability that this happens?

The probability of error

Take $p(x)$ and $q(x)$ of degree k

- If $(p(x), q(x)) \in$ EQUIV-POL, then the algorithm always gives the correct answer

- If $(p(x), q(x)) \notin$ EQUIV-POL, the algorithm can answer yes if it selected some $a \in \{1, \ldots, 100k\}$ such that $p(a) = q(a)$

  But then a is a root (a null-value) of the polynomial $r(x) = p(x) - q(x)$

The probability of error

We know that $r(x)$ is not a null-polynomial, and that its degree is no more than k

- Therefore $r(x)$ has at most k roots in $\mathbb{Q}$

We get:

$$\begin{aligned} \Pr(a \text{ is a root of } r(x)) &\leq \frac{k}{100k} \\ &= \frac{1}{100} \end{aligned}$$

Amplifying the probability that the algorithm is correct

The probability of error is bounded by $\frac{1}{100}$

- Is this good enough for you?

Amplifying the probability that the algorithm is correct

The probability of error is bounded by $\frac{1}{100}$

- Is this good enough for you?

Exercise

We can solve EQUIV-POL using $O(k)$ multiplications with probability of error bounded by $\frac{1}{100^{10}}$

Amplifying the probability that the algorithm is correct

The probability of error is bounded by $\frac{1}{100}$

- Is this good enough for you?

Exercise

We can solve EQUIV-POL using $O(k)$ multiplications with probability of error bounded by $\frac{1}{100^{10}}$

Good enough? (It is more probable that a meteorite crushes you)

Turing machines which account for randomness

Can we formalise the notion of randomness using TMs?

Can we define complexity classes which account for randomness?

Idea: Give a random number generator to a TM

- Clearly a random bit generator is enough
- Or a fair coin if you prefer

Probabilistic Turing Machines

Definition

A probabilistic TM is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$*:*

- Q *is a finite set of states*
- Σ *is a finite alphabet* $\vdash, \mathtt{B} \notin \Sigma$
- Γ *is the tape alphabet with* $\Sigma \cup \{\vdash, \mathtt{B}\} \subseteq \Gamma$
- $q_0 \in Q$ *is the initial state*
- $q_{accept}, q_{reject} \in Q$ *are the halting states*
- δ *is a partial function:*

$$\delta \quad : \quad Q \times \Gamma \times \{0, 1\} \to Q \times \Gamma \times \{\leftarrow, \square, \rightarrow\}$$

How do probabilistic TMs work?

The input to a probabilistic TM M consists of a string $w \in \Sigma^*$ and a string $s \in \{0,1\}^\omega$

- w is the input we want to accept of reject
- s is an infinite string of 0 and 1, which model our random bits/choices
- The intuition: s models a source of random bits (wait one slide)

The initial state:

- M has $\vdash w\mathrm{B}\cdots$ on its first tape, and $\vdash s$ on its second/"random" tape
- M is in state q_0
- Both reading heads of M are at position 1 of their tape

How do probabilistic TMs work?

At each step the machine is in some state q and its tape heads are in positions p_1 and p_2

- If the symbol in the position p_i $(i = 1, 2)$ is a_i and $\delta(q, a_1, a_2) = (q', b, X)$, then:
  - The machine writes the symbol b in the position p_1 of the **first** tape
  - We change the state from q to q'
  - The first head moves to $p_1 - 1$ if X is $\leftarrow$, and to $p_1 + 1$ if X is $\rightarrow$. If X is $\square$, then first head remains in the position p_1

How do probabilistic TMs work?

At each step the machine is in some state q and its tape heads are in positions p_1 and p_2

- If the symbol in the position p_i $(i = 1, 2)$ is a_i and $\delta(q, a_1, a_2) = (q', b, X)$, then:
  - The machine writes the symbol b in the position p_1 of the **first** tape
  - We change the state from q to q'
  - The first head moves to $p_1 - 1$ if X is $\leftarrow$, and to $p_1 + 1$ if X is $\rightarrow$. If X is $\square$, then first head remains in the position p_1
  - The second head moves to the position $p_2 + 1$

Running time of a probabilistic TM

The input to a probabilistic TM M with the alphabet Σ consists of two strings: $w \in \Sigma^*$ and $s \in \{0,1\}^\omega$

- We use the notation $M(w,s)$ to denote the input of M
- We say that $M(w,s)$ accepts, if M on input (w,s) halts in q_{accept}
  - We define that $M(w,s)$ rejects analogously

Running time of a probabilistic TM

The input to a probabilistic TM M with the alphabet Σ consists of two strings: $w \in \Sigma^*$ and $s \in \{0,1\}^\omega$

- We use the notation $M(w,s)$ to denote the input of M
- We say that $M(w,s)$ accepts, if M on input (w,s) halts in q_{accept}
  - We define that $M(w,s)$ rejects analogously

First assumption

We consider only probabilistic TMs M which halt on every input (w,s)

This is the same as when defining what an algorithm is.

Running time of a probabilistic TM

As before, a step of probabilistic TM M consists of executing a single instruction of the transition function

Running time of a probabilistic TM

As before, a step of probabilistic TM M consists of executing a single instruction of the transition function

- We define $time_M(w, s)$ as the number of steps executed by M on input (w, s)

Running time of a probabilistic TM

As before, a step of probabilistic TM M consists of executing a single instruction of the transition function

- We define $time_M(w, s)$ as the number of steps executed by M on input (w, s)

Second assumption

There is a function $f : \Sigma^* \to \mathbb{N}$ such that for all $w \in \Sigma^*$ and $s \in \{0, 1\}^\omega$:

$$time_M(w, s) \quad \leq \quad f(w)$$

Running time of a probabilistic TM

As before, a step of probabilistic TM M consists of executing a single instruction of the transition function

- We define $time_M(w, s)$ as the number of steps executed by M on input (w, s)

Second assumption

There is a function $f : \Sigma^* \to \mathbb{N}$ such that for all $w \in \Sigma^*$ and $s \in \{0, 1\}^\omega$:

$$time_M(w, s) \quad \leq \quad f(w)$$

That is, there is maximum number of random bits which can be used on input w, and this number depends only on w

Running time of a probabilistic TM

To study worst case complexity we define:

$$time_M(w) \quad = \quad \max\{time_M(w, s) \mid s \in \{0,1\}^\omega\}$$

Running time of a probabilistic TM

To study worst case complexity we define:

$$time_M(w) \quad = \quad \max\{time_M(w, s) \mid s \in \{0,1\}^\omega\}$$

With this we define the time complexity of a probabilistic TM M as the function t_M:

$$t_M(n) \quad = \quad \max\{time_M(w) \mid w \in \Sigma^* \text{ and } |w| = n\}$$

The probability that a probabilistic TM accepts

Third assumption

If for a probabilistic TM M with alphabet Σ it holds that $t_M(n) \leq g(n)$ for all $n \in \mathbb{N}$, then we assume that the inputs of M are of the form (w, s) where $w \in \Sigma^*$, $s \in \{0,1\}^*$ and $|s| = g(n)$.

Since the running time of M is bounded by $g(n)$, we can never use more than $g(n)$ bits of s on input w of length n.

The probability that a probabilistic TM accepts

Let M be a probabilistic TM with alphabet Σ, and let $t_M(n) \leq g(n)$ for all $n \in \mathbb{N}$.

Definition

For all $w \in \Sigma^*$ with $|w| = n$, the probability that M accepts w is defined as:

$$Pr[M \text{ accepts } w] = \frac{|\{s \in \{0,1\}^* \mid |s| = g(n) \text{ and } M(w,s) \text{ accepts}\}|}{2^{g(n)}}$$

Also, $\Pr[M \text{ rejects } w] = 1 - \Pr[M \text{ accepts } w]$

Why is this OK for any $t_M(n) \leq g(n) \leq g'(n)$?

Probabilistic complexity classes

The idea: polynomial time probabilistic machine which gives me the correct answer with high probability

Probabilistic complexity classes

The idea: polynomial time probabilistic machine which gives me the correct answer with high probability

Definition

Let L be a language over the alphabet Σ. We say that L is in the class BPP if there is a probabilistic TM M with $t_M(n) = O(n^k)$ and for each $w \in \Sigma^*$:

- *If $w \in L$, then $Pr[M \text{ accepts } w] \geq \frac{3}{4}$*
- *If $w \notin L$, then $Pr[M \text{ accepts } w] \leq \frac{1}{4}$*

Probabilistic complexity classes

The idea: polynomial time probabilistic machine which gives me the correct answer with high probability

Definition

Let L be a language over the alphabet Σ. We say that L is in the class BPP if there is a probabilistic TM M with $t_M(n) = O(n^k)$ and for each $w \in \Sigma^*$:

- *If $w \in L$, then $Pr[M \text{ accepts } w] \geq \frac{3}{4}$*
- *If $w \notin L$, then $Pr[M \text{ accepts } w] \leq \frac{1}{4}$*

Another way of saying this: $\Pr_{s \in \{0,1\}^{g(n)}}[M(x,s) = L(x)] \geq \frac{3}{4}$

(Here $L(x) = 1$ is $x \in L$, and $M(x,s) = 1$ if the machine accepts)

A basic observation

Theorem

$BPP = co\text{-}BPP$

A basic observation

Theorem

$BPP = co\text{-}BPP$

Exercise

Why is this true?

Amplification lemma

In the definition of BPP, the probability $\frac{3}{4}$ seems quite arbitrary. And this is true, we can make this much better.

Amplification lemma

In the definition of BPP, the probability $\frac{3}{4}$ seems quite arbitrary. And this is true, we can make this much better.

Error reduction for BPP

Let L be a language over Σ. If $L \in \text{BPP}$, then for each $\ell \in \mathbb{N}$, there is a probabilistic TM M with $t_M(n) = O(n^k)$ and for all $w \in \Sigma^*$:

- If $w \in L$, then $\Pr[M \text{ accepts } w] \geq 1 - \left(\frac{3}{4}\right)^\ell$
- If $w \notin L$, then $\Pr[M \text{ accepts } w] \leq \left(\frac{3}{4}\right)^\ell$

Idea: The same as for checking polynomial equality; just run the algorithm multiple times!

Proof of BPP error reduction

Since $L \in \text{BPP}$, there is a probabilistic TM M_1 with $t_{M_1}(n) = O(n^{k_1})$ and for all $w \in \Sigma^*$:

- If $w \in L$, then $\Pr[M_1 \text{ accepts } w] \geq \frac{3}{4}$
- If $w \notin L$, then $\Pr[M_1 \text{ accepts } w] \leq \frac{1}{4}$

Proof of BPP error reduction

Since $L \in \text{BPP}$, there is a probabilistic TM M_1 with $t_{M_1}(n) = O(n^{k_1})$ and for all $w \in \Sigma^*$:

- If $w \in L$, then $\Pr[M_1 \text{ accepts } w] \geq \frac{3}{4}$
- If $w \notin L$, then $\Pr[M_1 \text{ accepts } w] \leq \frac{1}{4}$

We use M_1 to construct the machine M from the lemma.

Proof of BPP error reduction

Consider the following algorithm which takes as input $w \in \Sigma^*$:

1. Randomly (and uniformly) select strings s_1, s_2, ... $s_{2\ell+1}$ in $\{0,1\}^*$ such that $|s_1| = |s_2| = \cdots = |s_{2\ell+1}| = t_{M_1}(|w|)$
2. Construct the set $A = \{i \in \{1, \ldots, 2\ell+1\} \mid M_1(w, s_i) \text{ accepts}\}$
3. If $|A| \geq \ell + 1$, then **accept**, otherwise **reject**

Proof of BPP error reduction

Consider the following algorithm which takes as input $w \in \Sigma^*$:

1. Randomly (and uniformly) select strings s_1, s_2, ... $s_{2\ell+1}$ in $\{0,1\}^*$ such that $|s_1| = |s_2| = \cdots = |s_{2\ell+1}| = t_{M_1}(|w|)$
2. Construct the set $A = \{i \in \{1, \ldots, 2\ell+1\} \mid M_1(w, s_i) \text{ accepts}\}$
3. If $|A| \geq \ell + 1$, then **accept**, otherwise **reject**

Assume that M is the probabilistic TM implementing this algorithm

Proof of BPP error reduction

Consider the following algorithm which takes as input $w \in \Sigma^*$:

1. Randomly (and uniformly) select strings s_1, s_2, ... $s_{2\ell+1}$ in $\{0,1\}^*$ such that $|s_1| = |s_2| = \cdots = |s_{2\ell+1}| = t_{M_1}(|w|)$
2. Construct the set $A = \{i \in \{1, \ldots, 2\ell+1\} \mid M_1(w, s_i) \text{ accepts}\}$
3. If $|A| \geq \ell + 1$, then **accept**, otherwise **reject**

Assume that M is the probabilistic TM implementing this algorithm

- Then $t_M(n)$ is $O(n^k)$ for some k, since $t_{M_1}(n)$ is $O(n^{k_1})$ for some k_1

Proof of BPP error reduction

Consider the following algorithm which takes as input $w \in \Sigma^*$:

1. Randomly (and uniformly) select strings s_1, s_2, ... $s_{2\ell+1}$ in $\{0,1\}^*$ such that $|s_1| = |s_2| = \cdots = |s_{2\ell+1}| = t_{M_1}(|w|)$
2. Construct the set $A = \{i \in \{1, \ldots, 2\ell+1\} \mid M_1(w, s_i) \text{ accepts}\}$
3. If $|A| \geq \ell + 1$, then **accept**, otherwise **reject**

Assume that M is the probabilistic TM implementing this algorithm

- Then $t_M(n)$ is $O(n^k)$ for some k, since $t_{M_1}(n)$ is $O(n^{k_1})$ for some k_1

Next we prove that M has the desired properties

Proof of BPP error reduction

Take first $w \in L$, and assume that

$$\Pr[M_1 \text{ accepts } w] \quad = \quad p$$

then by the assumption $p \geq \frac{3}{4}$

Proof of BPP error reduction

Take first $w \in L$, and assume that

$$\Pr[M_1 \text{ accepts } w] \quad = \quad p$$

then by the assumption $p \geq \frac{3}{4}$

Furthermore, assume that $p < 1$

- If $p = 1$ then $\Pr[M \text{ accepts } w] = 1 \geq 1 - (\frac{3}{4})^{\ell}$. Why?

Proof of BPP error reduction

Take first $w \in L$, and assume that

$$\Pr[M_1 \text{ accepts } w] \quad = \quad p$$

then by the assumption $p \geq \frac{3}{4}$

Furthermore, assume that $p < 1$

- If $p = 1$ then $\Pr[M \text{ accepts } w] = 1 \geq 1 - (\frac{3}{4})^{\ell}$. Why?

Therefore we have that:

$$\Pr[M \text{ rejects } w] \quad = \quad \sum_{i=0}^{\ell} \binom{2\ell+1}{i} p^i (1-p)^{2\ell+1-i}$$

Proof of BPP error reduction

Since $\frac{3}{4} \leq p < 1$, it follows that $1 \leq \frac{p}{1-p}$

Therefore for $i \in \{0, \ldots, \ell\}$ it holds:

$$\begin{aligned} p^i(1-p)^{2\ell+1-i} &\leq p^i(1-p)^{2\ell+1-i}\left(\frac{p}{1-p}\right)^{\ell-i} \\ &= p^\ell(1-p)^{\ell+1} \end{aligned}$$

Proof of BPP error reduction

Therefore:

$$\begin{aligned}
\sum_{i=0}^{\ell} \binom{2\ell+1}{i} p^i (1-p)^{2\ell+1-i} &\leq \sum_{i=0}^{\ell} \binom{2\ell+1}{i} p^\ell (1-p)^{\ell+1} \\
&= p^\ell (1-p)^{\ell+1} \sum_{i=0}^{\ell} \binom{2\ell+1}{i} \\
&\leq p^\ell (1-p)^{\ell+1} 2^{2\ell+1}
\end{aligned}$$

Proof of BPP error reduction

Therefore:

$$\begin{aligned}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}p^i(1-p)^{2\ell+1-i} &\leq \sum_{i=0}^{\ell}\binom{2\ell+1}{i}p^\ell(1-p)^{\ell+1}\\ &= p^\ell(1-p)^{\ell+1}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}\\ &\leq p^\ell(1-p)^{\ell+1}2^{2\ell+1}\end{aligned}$$

The maximum value of $f(x) = x(1-x)$ in the interval $[\frac{3}{4}, 1)$ is achieved in $x = \frac{3}{4}$

Proof of BPP error reduction

Therefore:

$$\begin{aligned}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}p^i(1-p)^{2\ell+1-i} &\leq \sum_{i=0}^{\ell}\binom{2\ell+1}{i}p^{\ell}(1-p)^{\ell+1}\\ &= p^{\ell}(1-p)^{\ell+1}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}\\ &\leq p^{\ell}(1-p)^{\ell+1}2^{2\ell+1}\end{aligned}$$

The maximum value of $f(x) = x(1-x)$ in the interval $[\frac{3}{4}, 1)$ is achieved in $x = \frac{3}{4}$

- We conclude that $p(1-p) \leq \frac{3}{4} \cdot \frac{1}{4}$

Proof of BPP error reduction

It now follows that:

$$
\begin{aligned}
\sum_{i=0}^{\ell} \binom{2\ell+1}{i} p^i (1-p)^{2\ell+1-i} &\leq p^\ell (1-p)^{\ell+1} 2^{2\ell+1} \\
&= (p(1-p))^\ell (1-p) 2^{2\ell+1} \\
&\leq \left(\frac{3}{4} \cdot \frac{1}{4}\right)^\ell \cdot \frac{1}{4} 2^{2\ell+1} \\
&= \frac{3^\ell}{4^{2\ell+1}} 2^{2\ell+1} \\
&= 2 \frac{3^\ell}{4^{2\ell+1}} 4^\ell \\
&= \frac{1}{2} \cdot \frac{3^\ell}{4^\ell} \\
&< \left(\frac{3}{4}\right)^\ell
\end{aligned}
$$

Proof of BPP error reduction

So we can conclude that:

$$\begin{aligned} \Pr[M \text{ rejects } w] &= \sum_{i=0}^{\ell} \binom{2\ell+1}{i} p^i (1-p)^{2\ell+1-i} \\ &< \left(\frac{3}{4}\right)^{\ell} \end{aligned}$$

Which gives us:

$$\Pr[M \text{ accepts } w] \geq 1 - \left(\frac{3}{4}\right)^{\ell}$$

Proof of BPP error reduction

Consider now the case when $w \notin L$, and let

$$\Pr[M_1 \text{ accepts } w] \quad = \quad q$$

where $q \leq \frac{1}{4}$

Proof of BPP error reduction

Consider now the case when $w \notin L$, and let

$$\Pr[M_1 \text{ accepts } w] \quad = \quad q$$

where $q \leq \frac{1}{4}$

Furthermore, assume that $0 < q$

- If $q = 0$ then we have that $\Pr[M \text{ accepts } w] = 0 \leq (\frac{3}{4})^{\ell}$. Why?

Proof of BPP error reduction

Consider now the case when $w \notin L$, and let

$$\Pr[M_1 \text{ accepts } w] \quad = \quad q$$

where $q \leq \frac{1}{4}$

Furthermore, assume that $0 < q$

- If $q = 0$ then we have that $\Pr[M \text{ accepts } w] = 0 \leq (\frac{3}{4})^{\ell}$. Why?

Same as in the previous case, we now have:

$$\Pr[M \text{ accepts } w] \quad = \quad \sum_{i=0}^{\ell} \binom{2\ell+1}{i} (1-q)^i q^{2\ell+1-i}$$

Proof of BPP error reduction

Since $0 < q \leq \frac{1}{4}$, we get that $1 \leq \frac{1-q}{q}$

So for all $i \in \{0, \ldots, \ell\}$ it holds:

$$\begin{aligned}(1-q)^i q^{2\ell+1-i} &\leq (1-q)^i q^{2\ell+1-i}\left(\frac{1-q}{q}\right)^{\ell-i} \\ &= (1-q)^\ell q^{\ell+1}\end{aligned}$$

Proof of BPP error reduction

Therefore:

$$\begin{aligned}
\sum_{i=0}^{\ell} \binom{2\ell+1}{i} (1-q)^i q^{2\ell+1-i} &\leq \sum_{i=0}^{\ell} \binom{2\ell+1}{i} (1-q)^{\ell} q^{\ell+1} \\
&= (1-q)^{\ell} q^{\ell+1} \sum_{i=0}^{\ell} \binom{2\ell+1}{i} \\
&\leq (1-q)^{\ell} q^{\ell+1} 2^{2\ell+1}
\end{aligned}$$

Proof of BPP error reduction

Therefore:

$$\begin{aligned}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}(1-q)^i q^{2\ell+1-i} &\leq \sum_{i=0}^{\ell}\binom{2\ell+1}{i}(1-q)^\ell q^{\ell+1}\\ &= (1-q)^\ell q^{\ell+1}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}\\ &\leq (1-q)^\ell q^{\ell+1}2^{2\ell+1}\end{aligned}$$

The maximum of the function $f(x) = x(1-x)$ in the interval $(0, \frac{1}{4}]$ is achieved in $x = \frac{1}{4}$

Proof of BPP error reduction

Therefore:

$$\begin{aligned}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}(1-q)^i q^{2\ell+1-i} &\leq \sum_{i=0}^{\ell}\binom{2\ell+1}{i}(1-q)^\ell q^{\ell+1} \\ &= (1-q)^\ell q^{\ell+1}\sum_{i=0}^{\ell}\binom{2\ell+1}{i} \\ &\leq (1-q)^\ell q^{\ell+1}2^{2\ell+1}\end{aligned}$$

The maximum of the function $f(x) = x(1-x)$ in the interval $(0, \frac{1}{4}]$ is achieved in $x = \frac{1}{4}$

- We can conclude that $(1-q)q \leq \frac{3}{4} \cdot \frac{1}{4}$

Proof of BPP error reduction

From this we get:

$$\begin{aligned}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}(1-q)^i q^{2\ell+1-i} &\leq (1-q)^\ell q^{\ell+1}2^{2\ell+1}\\ &= ((1-q)q)^\ell q 2^{2\ell+1}\\ &\leq \left(\frac{3}{4}\cdot\frac{1}{4}\right)^\ell \cdot \frac{1}{4}2^{2\ell+1}\end{aligned}$$

Proof of BPP error reduction

From this we get:

$$\begin{aligned}\sum_{i=0}^{\ell}\binom{2\ell+1}{i}(1-q)^i q^{2\ell+1-i} &\leq (1-q)^\ell q^{\ell+1}2^{2\ell+1}\\ &= ((1-q)q)^\ell q 2^{2\ell+1}\\ &\leq \left(\frac{3}{4}\cdot\frac{1}{4}\right)^\ell \cdot \frac{1}{4}2^{2\ell+1}\end{aligned}$$

As in the case when $x \in L$ we get that:

$$\sum_{i=0}^{\ell}\binom{2\ell+1}{i}(1-q)^i q^{2\ell+1-i} < \left(\frac{3}{4}\right)^\ell$$

Proof of BPP error reduction

And finally, we can conclude that:

$$\begin{aligned} \Pr[M \text{ accepts } w] &= \sum_{i=0}^{\ell} \binom{2\ell+1}{i} (1-q)^i q^{2\ell+1-i} \\ &\leq \left(\frac{3}{4}\right)^{\ell} \end{aligned}$$

□

Strong amplification lemma

In fact, the amplification lemma can be made much stronger:

Lemma (Exponential error reduction for BPP)

Take any polynomial $poly(n)$. Then for all $L \in BPP$ there is a probabilistic polynomial time TM M such that:

- *If $w \in L$, then $Pr[M \text{ accepts } w] \geq 1 - \frac{1}{2^{poly(n)}}$*
- *If $w \notin L$, then $Pr[M \text{ accepts } w] \leq \frac{1}{2^{poly(n)}}$*

Strong amplification lemma: the proof

The proof follows from the weak version. We just have to find the correct ℓ for each n, and run the algorithm from that lemma ℓ times.

If this number is polynomial in n we are good

So fix some n

We want:

$$(\frac{3}{4})^{\ell} \leq 2^{-poly(n)}$$

Strong amplification lemma: the proof

$$(\frac{3}{4})^{\ell} \leq 2^{-poly(n)} \quad \Big/ log_2$$

$$\ell \cdot log_2(0.75) \leq -poly(n) \quad \Big/ : log_2(0.75) \quad (< 0)$$

$$\ell \geq -\frac{poly(n)}{log_2(0.75)}$$

$$\ell \geq \approx 2.5 poly(n)$$

Therefore, if we take $\ell = 3poly(n)$ we are good

Strong amplification lemma: the proof

The result we will actually use is that this holds for $poly(n) = n^2$

But you can prove by an easy induction on n that:

$$(\frac{3}{4})^{3n^2} \leq 2^{-n^2}$$

Relating BPP to other complexity classes

With the amplification lemma we can easily obtain this result:

Theorem (Adleman, 1978)

$BPP \subseteq P/poly$.

Proof: Take any $L \in$ BPP. By the strong error reduction, we know that there is a polynomial time probabilistic TM M such that:

- If $w \in L$, then $\Pr[M \text{ accepts } w] \geq 1 - \frac{1}{2^{n^2}}$
- If $w \notin L$, then $\Pr[M \text{ accepts } w] \leq \frac{1}{2^{n^2}}$

Small circuits for BPP

Since $t_M(n) \leq p(n)$, for some polynomial $p(n)$:

- We can assume that the random string s for M is of length $p(n)$

So we have that:

For all $w \in \Sigma^*$ with $|w| = n$, the probability that M accepts w is defined as:

$$\Pr[M \text{ accepts } w] \quad = \quad \frac{|\{s \in \{0,1\}^* \mid |s| = p(n) \text{ and } M(w,s) \text{ accepts}\}|}{2^{p(n)}}$$

Small circuits for BPP

Fix any $x \in \{0,1\}^n$

By our assumption:

$$\Pr_{s \in \{0,1\}^{p(n)}}[M(x,s) \neq L(x)] \leq \frac{1}{2^{n^2}}$$

Therefore the number of $s \in \{0,1\}^{p(n)}$ such that $M(x,s) \neq L(x)$:

- Is at most $2^{p(n)} \cdot 2^{-n^2}$

So for all $x \in \{0,1\}^n$ the number of random bit-strings s which make the machine output the wrong answer is at most:

$$2^n \cdot 2^{p(n)} \cdot 2^{-n^2} = 2^{p(n)} \cdot 2^{-n} < 2^{p(n)}$$

Small circuits for BPP

Therefore, there is a string r_n of length $p(n)$ such that:

- For all $x \in \{0,1\}^n$ we have that $M(x, r_n) = L(x)$
- That is, the machine gives the correct answer

Small circuits for BPP

Therefore, there is a string r_n of length $p(n)$ such that:

- For all $x \in \{0,1\}^n$ we have that $M(x, r_n) = L(x)$
- That is, the machine gives the correct answer

This holds for all n, so we have a sequence $\{r_n\}_n$ of strings such that $M(x, r_{|x|}) = L(x)$, for any x

Small circuits for BPP

Therefore, there is a string r_n of length $p(n)$ such that:

- For all $x \in \{0,1\}^n$ we have that $M(x, r_n) = L(x)$
- That is, the machine gives the correct answer

This holds for all n, so we have a sequence $\{r_n\}_n$ of strings such that $M(x, r_{|x|}) = L(x)$, for any x

A polytime machine with polynomial advice string for L:

- Use r_n as the advice string for the machine N
- Run M on input $(x, r_{|x|})$
- Enjoy life!

BPP and PH

Theorem (Gács-Sipser-Lautemann)

$BPP \subseteq \Sigma_2^P \cap \Pi_2^P$

BPP and PH

Theorem (Gács-Sipser-Lautemann)

$BPP \subseteq \Sigma_2^P \cap \Pi_2^P$

Since we know that BPP = co-BPP, we only need to prove that BPP $\subseteq \Sigma_2^P$

So take any $A \in$ BPP

We need to give a Σ_2^P algorithm for A

A quick reminder

Proposition

Let L be a language over Σ. Then L is in Σ_2^P if and only if there is a PTIME decidable language B and a polynomial $q(n)$ such that for all $w \in \Sigma^*$:

$$w \in L \text{ if and only if}$$
$$(\exists v_1 \in \Sigma^*, |v_1| = q(|w|))(\forall v_2 \in \Sigma^*, |v_2| = q(|w|)) : (w, v_1, v_2) \in B$$

Proof of BPP $\subseteq \Sigma_2^P$

Since $A \in$ BPP by the strong amplification lemma:

- There is a probabilistic TM M
- The running time of M is $p(n)$, with p a polynomial
- M uses $p(n)$ bits on input of length n
- Why can I assume the last two simultaneously?

- And the following error bounds hold:
  - $x \in A$ then $\Pr[M(x) \text{ accepts}] \geq 1 - 2^{-|x|}$
  - $x \notin A$ then $\Pr[M(x) \text{ accepts}] \leq 2^{-|x|}$

Proof of BPP $\subseteq \Sigma_2^P$

To make thing more explicit:

Since $A \in$ BPP by the strong amplification lemma:

- There is a probabilistic TM M
- The running time of M is $p(n)$, with p a polynomial
- M uses $p(n)$ bits on input of length n
- And the following error bounds hold:
  - $x \in A$ then $\Pr_{y \in \{0,1\}^{p(|x|)}}[M(x, y) \text{ accepts}] \geq 1 - 2^{-|x|}$
  - $x \notin A$ then $\Pr_{y \in \{0,1\}^{p(|x|)}}[M(x, y) \text{ accepts}] \leq 2^{-|x|}$

Proof of BPP $\subseteq \Sigma_2^P$

Fix any x of length n and let $m = p(n)$

$$A_x = \{y \in \{0,1\}^m \mid M(x,y) \text{ accepts } \}$$

$$R_x = \{y \in \{0,1\}^m \mid M(x,y) \text{ rejects } \}$$

Note that:

$$R_x = \{0,1\}^m - A_x$$

Proof of BPP $\subseteq \Sigma_2^P$

$$A_x = \{y \in \{0,1\}^m \mid M(x,y) \text{ accepts } \}$$

It holds that:

If $x \in A$: $\quad |A_x| \geq 2^m - 2^{m-n}$ and $|R_x| \leq 2^{m-n}$

If $x \notin A$: $\quad |R_x| \geq 2^m - 2^{m-n}$ and $|A_x| \leq 2^{m-n}$

Proof of BPP $\subseteq \Sigma_2^P$

Notation

We use $s \in \{0,1\}^m$ to indicate that $s \in \{0,1\}^*$ and $|s| = m$

Given a and b in $\{0,1\}$, the operation $a \oplus b$ is defined as $(a + b) \bmod 2$

- That is, $\oplus$ is XOR

For $x, y \in \{0,1\}^m$ with $x = a_1 a_2 \cdots a_m$ and $y = b_1 b_2 \cdots b_m$, the operation $x \oplus y$ gives as the result the following string in $\{0,1\}^m$:

$$(a_1 \oplus b_1)(a_2 \oplus b_2) \cdots (a_m \oplus b_m)$$

We will use the fact that $a \oplus a = 0^m$

Proof of BPP $\subseteq \Sigma_2^P$

The following lemma is key to our algorithm:

Lemma

The string $x \in A$ if and only if there exist strings $z_1 \ldots, z_m$, each of length m, such that

$$\{y \oplus z_j \mid 1 \leq j \leq m, y \in A_x\} \quad = \quad \{0,1\}^m,$$

where $|x| = n$, and $m = p(n)$ as above.

The idea is that for $x \in A$, the set A_x is so big that just a few shifts of the form $y \mapsto y \oplus z$ can cover all of $\{0,1\}^m$ (and the opposite for $x \notin A$)

Proof of BPP $\subseteq \Sigma_2^P$

From the lemma we get the following characterisation of A:

A string x of length n is in A iff (here $m = p(n)$):

1. $\exists z_1 \in \{0,1\}^m \cdots \exists z_m \in \{0,1\}^m$
2. $\forall w \in \{0,1\}^m$
3. $w = y \oplus z_j$, for some $1 \leq j \leq m$ and $y \in A_x$

Proof of BPP $\subseteq \Sigma_2^P$

From the lemma we get the following characterisation of A:

A string x of length n is in A iff (here $m = p(n)$):

1. $\exists z_1 \in \{0,1\}^m \cdots \exists z_m \in \{0,1\}^m$
2. $\forall w \in \{0,1\}^m$
3. $w = y \oplus z_j$, for some $1 \leq j \leq m$ and $y \in A_x$

The last property is saying that $w \oplus z_j \in A_x$ (since $y = w \oplus z_j$)

- So we just run $M(x, w \oplus z_j)$ for $j = 1 \ldots m$ (which is PTIME)

Therefore BPP $\subseteq \Sigma_2^P$

Proof of the lemma

To finish the proof we only need to prove the lemma

[$\Leftarrow$] Assume that $x \notin A$

We know that $|A_x| \leq 2^{m-n}$

So take **any** $z_1, \ldots, z_m$

Since the function $y \mapsto y \oplus z$ is injective:

$$\{y \oplus z_j \mid 1 \leq j \leq m, y \in A_x\} = \bigcup_{j=1}^{m} \{y \oplus z_j \mid y \in A_x\}$$

Proof of the lemma

$$
\begin{aligned}
|\{y \oplus z_j \mid 1 \leq j \leq m, y \in A_x\}| &\leq \sum_{j=1}^{m} \{y \oplus z_j \mid y \in A_x\} \\
&= \sum_{j=1}^{m} |A_x| \\
&\leq m2^{m-n} \\
&< 2^m
\end{aligned}
$$

(for n big enough)

Therefore $\{y \oplus z_j \mid 1 \leq j \leq m, y \in A_x\} \neq \{0,1\}^m$

So we are done with $[\Leftarrow]$

Proof of the lemma

$[\Rightarrow]$ Assume that $x \in A$

Then $|R_x| \leq 2^{m-n}$

We call a a sequence $z_1, \ldots, z_m$ **bad** (for x) if:

- There is $w \in \{0,1\}^m$ such that

$$\{w \oplus z_j \mid 1 \leq j \leq m\} \subseteq R_x$$

A sequence that is not bad is called **good** (for x)

We wish to prove that there is some good sequence for x

Proof of the lemma

How many bad sequences are there?

A bad sequence is defined by:

- A subset of R_x of size m
- And a string $w \in \{0,1\}^m$
- (This follows since we want $w \oplus z_j \in R_x$)

Since $|R_x| \leq 2^{m-n}$, then R_x has $\leq (2^{m-n})^m$ subsets of size m

So the number of bad sequences is at most:

$$2^m \cdot (2^{m-n})^m \leq 2^{m(m-n+1)} < 2^{m^2}$$

Proof of the lemma

The number of bad sequences is at most:

$$2^m \cdot (2^{m-n})^m \leq 2^{m(m-n+1)} < 2^{m^2}$$

On the other hand, the total number of (any) sequences is 2^{m^2}

So there must exist at least one good sequence