April 28, 2013

This file contains the scribes and exercises done by the students of the Ph.D. core class *Theory of Computation* at Northeastern University in Spring 2013, taught by Emanuele Viola.

Disclaimer: This file has not been edited by the instructor, and it contains mistakes.

Lecture 1

1.1 Summary

During this lecture we continued going through "Think like the pros" covering the topics of "Proof by Contradiction" and "Quantifiers as Games". We explored multiple examples of proof by contradiction such as irrationality of $\sqrt{2}$ and tiling of mutilated chessboards. We also covered some examples in order of growth and sets and functions. □

1.2 Proof of infinite primes

Claim 1. There exist infinitely many prime numbers. Recall that x is prime if x is only divisible by 1 or x.

Proof. By contradiction. Suppose there exist only finitely many primes. Call them $p_1, \ldots, p_k$, for some k. Consider the value:

$$q = \prod_{i \leq k} p_i + 1$$

This value must be prime, contradicting the supposition. Indeed, suppose q is not prime; then there must exist some $p_i > 1$ dividing q. But this is impossible, because p_i divides $q - 1$ and $p_i > 1$. □

1.3 Board tilings

Consider an 8×8 board.

Question: Can you tile (completely cover the board with no overlap) the board with 2×1 pieces?
Answer: Yes, by placing four tiles in each row. □

Question: Remove 2 opposite corners. Can you tile with 2×1 pieces?
Answer: No.

Proof. Color the board like a chess board, in black and white.

i. Note that we removed two squares of the same color, say white

ii. Also note that each 2×1 piece has one black and one white square Assume for contradiction that the board can be tiled.

Then by ii. you have the same number of black and white squares, but by i. there are fewer white than black squares. This is a contradiction. □

Exercises for PhD Core Theory of Computation (S '13)

Exercise 1. Prove that $\sqrt[4]{2}$ is irrational (using only what has been seen in class).

Solution:

Claim 2. $\nexists a, b \in \mathbb{Z} : 2a^4 = b^4$

Proof. Suppose, for later contradiction, that

$$\text{i. } \exists a, b \in \mathbb{Z} : 2a^4 = b^4.$$

Now let $c := a^2$ and $d := b^2$. Since a and b are integers, c and d must also be integers. This allows us to rewrite i. as:

$$\text{ii. } \exists c, d \in \mathbb{Z} : 2c^2 = b^2.$$

However, if this is true than $\sqrt{2}$ is rational, which has been disproven in class. Therefore, ii. is contradicted and i. cannot be true. □

End of solution.

Exercise 2. Prove that:

1. On an 8×8 board, if you remove any corner you cannot tile the board with 3×1 pieces.
   Solution: Let's color all squares with the three colors white, black and red like the following picture:

w	b	r	w	b	r	w	b
b	r	w	b	r	w	b	r
r	w	b	r	w	b	r	w
w	b	r	w	b	r	w	b
b	r	w	b	r	w	b	r
r	w	b	r	w	b	r	w
w	b	r	w	b	r	w	b
b	r	w	b	r	w	b	r

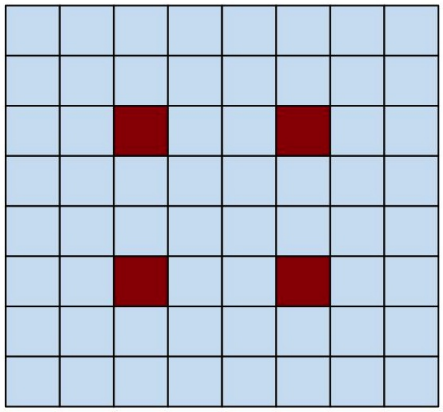
   When we place a straight tromino on the board it covers squares of all three colors. Therefore, after placing all tromino pieces they must cover 21 red, 21 white and 21 black squares. As you can see from the figure, the board has 21 red squares, 21 white squares, and 22 black squares. This means we should remove a black square in order to be able to tile the board with straight tromino pieces. (Or remove a white or red square if we colored the board in a different order).

Also, the final solution should still work if the board is reflected along its horizontal or vertical axes, so after these transformations the removed square has to still be located in a black square.

Since the black corners do not map on black squares when reflected therefore we can't remove any of the corners and tile the board with straight trominoes.

End of solution.

2. On an 8×8 board, show that there exists a square such that, if removed, you can tile the remaining squares with 3×1 pieces.
Solution: There are 4 squares on the board which satisfy both criteria from the solution to the previous part. The squares c3, c6, f3, and f6 are black and will be placed on black squares after any reflection.

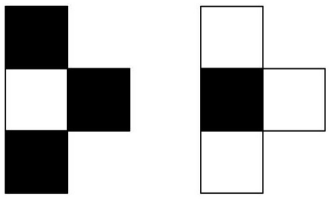


If we remove any of these black squares we will have 21 white squares, 21 red squares and 21 black squares which can be tiled with 21 trominoes.

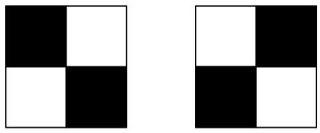
1	1	1	2	2	2	3	4
5	5	5	6	6	6	3	4
7	8	9	10	11	e	3	4
7	8	9	10	11	12	12	12
7	8	9	10	11	13	13	13
14	14	14	17	18	19	20	21
15	15	15	17	18	19	20	21
16	16	16	17	18	19	20	21

End of solution.

3. An 8×8 board cannot be tiled with 15 pieces (of 4 squares, shaped like a T) and one 2×2 square.
   Solution: Consider coloring the board with the colors black and white like a chessboard. We have two possible kinds of T-tetronimoes after coloring:



   And we have two kind of squares after coloring:



   Both the number of black squares and the number of white squares in our board are 32. We are using one 4×4 square which contains 2 black cells and 2 white cells, so we are left with 30 black cells and 30 white cells which should be covered with the 15 T-tetronimo pieces. Each T-tetronimo contains either 3 black cells and 1 white cell or 3 white cells and 1 one black cell, and since we should cover the same number of black and white cells we should have the same number of T-tetronimoes with 3 black cells as T-tetronimoes with 3 white cells. But we are using 15 T-tetronimoes, which is an odd number, therefore it is not possible to cover the remaining cells with them.
   End of solution.

Exercise 3. Prove the following.

1. Prove that for every integer $k > 0$, $\binom{n}{k} = \Theta(n^k)$.
   Solution: By definition, $\binom{n}{k} = \Theta(n^k) \Leftrightarrow \binom{n}{k} = O(n^k) \wedge \binom{n}{k} = \Omega(n^k)$. I will prove these separately.

   Claim 3. $\forall k > 0, \binom{n}{k} = O(n^k)$. That is, $\forall k > 0 \exists c, n_0 \forall n \geq n_0, \binom{n}{k} \leq cn^k$.

   Proof. Let $c = 1, n_0 = 1$. This gives us $\forall k > 0, n \geq 1 : \binom{n}{k} \leq n^k$. To see that this is true, we can use the hint:
   $\binom{n}{k} \leq (\frac{en}{k})^k$

$\leq (\frac{e}{k})^k n^k$ the first sentence is constant

$\leq c_1 n^k$

□

Claim 4. $\binom{n}{k} = \Omega(n^k)$. That is, $\forall k > 0 \exists c, n_0 \forall n \geq n_0, \binom{n}{k} \geq cn^k$.

Proof. According to the hint:
$\binom{n}{k(n)} \geq (\frac{n}{k})^k$

$\geq (\frac{1}{k})^k n^k$ the fisrt sentence is a constant

$\geq c_2 n^k$

□

End of solution.

2. Let $k : \mathbb{N} \to \mathbb{N}$ be a function. Suppose that $k(n)$ is $\omega(1)$. Prove that $\binom{n}{k(n)} = o(n^{k(n)})$.
Solution: Since $k(n) = \omega(1)$, we know that $\forall c > 0 \exists n_0 : \forall n \in \mathbb{N}, n \geq n_0 : k(n) > c$. In other words, if $P_\forall$ chooses any constant c then $P_\exists$ can choose n_0 so that the value of $k(n)$ will be greater than c for all $n > n_0$. This means that $k(n)$ grows in faster than constant time as n increases. This leads us toward the proof.

Claim 5. $\forall c > 0 \exists n_0 : \forall n \in \mathbb{N}, n \geq n_0 : \binom{n}{k(n)} < cn^{k(n)}$

Proof. According to hint:
$\binom{n}{k(n)} \leq (\frac{en}{k(n)})^{k(n)}$

$< (\frac{en}{c})^{k(n)}$

$< (\frac{e}{c})^{k(n)} n^{k(n)}$ if we choose c bigger than e then the first sentence is a constant

$< c_1 n^{k(n)}$

□

End of solution.

3. Let $k : \mathbb{N} \to \mathbb{N}$ be a function. Suppose $k(n)$ is both $\omega(1)$ and $O(\sqrt{n})$. Prove that $\binom{n}{k(n)} = n^{\omega(1)}$.
Solution: Since $k(n) = \omega(1)$, we know that $\forall c > 0 \exists n_0 : \forall n \in \mathbb{N}, n \geq n_0 : k(n) > c$. We also know $k(n) = O(\sqrt{n})$ which means $\forall c > 0 \exists n_0 : \forall n \in \mathbb{N}, n \geq n_0 : k(n) < c\sqrt{n}$

Claim 6. $\forall c_1, c > 0 \exists n_0 : \forall n \in \mathbb{N}, n \geq n_0 : \binom{n}{k(n)} > c_1 n^c$

Proof. We can follow:
$\binom{n}{k(n)} \geq (n/k)^k > (n/\sqrt{n})^k > (c_1\sqrt{n})^c > c_1^c n^{c/2}$ $\square$

End of solution.

Hint: use that for integers $n, k > 0 : (n/k)^k \leq \binom{n}{k} \leq (en/k)^k$, where $e < 2.7183$.

Exercise 4. Let $a = a_0, a_1, a_2, \ldots$ be a sequence of integers. Let us write $a \to \infty$ if $\forall t$, for sufficiently large $i, a_i \geq t$ (this is a definition). Prove that, for any two sequences a and b, if $a \to \infty$ and $b \to \infty$, then the sequence $c = c_0, c_1, c_2, \ldots$ defined as $c_i := a_{b_i}$ also $\to \infty$.

Solution:

Claim 7. $\forall t, \exists i_0 \forall i > i_0, c_i > t$

Proof. We are given the following three facts:

i. $\forall t, \exists i_0 \forall i > i_0, a_i > t$
ii. $\forall t, \exists i_0 \forall i > i_0, b_i > t$
iii. $c_i = a_{b_i}$

If we replace i with b_i in i., we get the following:

$$\forall t, \exists b_0 \forall b_i > b_0, a_{b_i} > t$$

From ii., we know that all sufficiently large b are greater than any given b_0. Therefore, we know that there exists some b_0 such that for all $b_i > b_0, a_{b_i} > t$. Since $a_{b_i} = c_i$, this tells us that the claim is true and that $c \to \infty$. $\square$

End of solution.

Exercise 5. Let $A = \{x|a(x)\}, B = \{x|b(x)\}$ be sets. Prove that

$$A = B \Leftrightarrow \left(A \cap \bar{B}\right) \cup \left(\bar{A} \cap B\right) = \varnothing$$

Solution: Given the rules for set equality in the slides, we know that $A = B$ precisely when $a(x) \Leftrightarrow b(x)$. This is also the case for the expression on the right, as can be shown by converting the claims from set notation into logical notation:

$$\begin{aligned} &\left(A \cap \bar{B}\right) \cup \left(\bar{A} \cap B\right) = \varnothing \\ \to &\{x|\, (a(x) \wedge \neg b(x)) \vee (\neg a(x) \wedge b(x))\} = \varnothing \end{aligned}$$

This merged set is the disjunction of two sets: any elements x for which $a(x)$ is true and $b(x)$ is false, and any elements x for which $a(x)$ is false and $b(x)$ is true. In other words, it is the set of all elements for which $a(x)$ and $b(x)$ give different outputs. This merged set is empty if and only if $a(x) \Leftrightarrow b(x)$, which is equivalent to $A = B$.

End of solution.

Lecture 2

We continued reading "Think Like the Pros" from Induction to Counting.

2.4 Proof of the Sunflower Theorem

Definition 8. We say that sets $A_1, A_2, \ldots, A_k$ are a sunflower of size k if $A_i \cap A_j$ is equal, for any $i \neq j$.

Theorem 9. Any family of $> s!(k-1)^s$ distinct sets of size s contains a sunflower of size k.

Proof. By induction on s:

Base case:

$s = 1$. We have $> (k-1)$ sets of size 1. Recall that they are distinct. Any k of these sets form a sunflower $A_1, \ldots, A_k$ with $A_i \cap A_j = \varnothing$, for any $i \neq j$.

Induction step:

Pick as many disjoint sets as you can. Call them $D_1, D_2, \ldots, D_t$.

If $t \geq k, D_1, D_2, \ldots, D_k$ is a sunflower of size k, with $D_i \cap D_j = \varnothing$ for any $i \neq j$.

If $t < k$, first observe that the total number of elements in the disjoint sets has the upper bound

$$\left| \bigcup_{i \leq t} D_i \right| \leq (k-1)s,$$

because each set has s elements, the disjoint sets all have different elements, and there are at most $k-1$ of these sets. Next, observe that any set in the family intersects some D_i. This is true because each set D_i intersects itself, and each set which is not some D_i must intersect at least one of them or else it would be disjoint from all of them and be one of the D_i sets. Combining these observations, and using the pigeonhole principle, there must exist some element x that belongs to

$$\frac{\#\text{ total sets}}{\#\text{ elements in disjoint sets}} > \frac{s!(k-1)^s}{s(k-1)} = (s-1)!(k-1)^{s-1}$$

sets. Let $A_1, A_2, \ldots, A_u$ be these sets. By the inductive hypothesis, $A_1 - \{x\}, A_2 - \{x\}, \ldots, A_u - \{x\}$ contains a sunflower of size k. Since $A_1, A_2, \ldots, A_u$ all contain x their intersections are still equal, so they also contain a sunflower of size k. □

Exercises for PhD Core Theory of Computation (S '13)

Exercise 6. Prove by induction that $1^3 + 2^3 + 3^3 + \cdots + n^3 = (1 + 2 + 3 + \cdots + n)^2$

Solution: By induction on n.

Base Case: $n = 1, 1^3 = 1^2$

Inductive Step: We assume that $1^3 + 2^3 + \cdots + (n-1)^3 = (1 + 2 + 3 + \cdots + n - 1)^2$. It then follows that:

$$\begin{aligned}
1^3 + 2^3 + 3^3 + \cdots + n^3 &= 1^3 + 2^3 + \cdots + (n-1)^3 + n^3 \\
&= (1 + 2 + 3 + \cdots + n - 1)^2 + n^3 && \text{by inductive hypothesis} \\
&= \left(\frac{n^2 - n}{2}\right)^2 + n^3 && \text{sum of the first } n-1 \text{ numbers} \\
&= \frac{n^4 - 2n^3 + n^2}{4} + n^3 \\
&= \frac{n^4 + 2n^3 + n^2}{4} \\
&= \left(\frac{n^2 + n}{2}\right)^2 \\
&= (1 + 2 + 3 + \cdots + n)^2 && \square
\end{aligned}$$

End of solution.

Exercise 7. Prove that any $2^n \times 2^n$ board with any one square removed can be tiled with L-trominos.

Solution: By induction on n.
Base Case: $n = 1$, there are 4 squares in total, so no matter which one is removed, the rest can be tiled with an L-tromino.
Inductive Step: We assume the statement holds for $k = n - 1$. When $k = n$ we can split the board into 4 equal-sized pieces, each having $2^{n-1} \times 2^{n-1}$ squares. By inductive hypothesis, the piece with the removed square can be tiled.

Now put one L-tromino in the center of the $2^n \times 2^n$ board so that each square of the L-tromino touches one square of the remaining three $2^{n-1} \times 2^{n-1}$ boards. We remove these three touched squares. By inductive hypothesis, the three remaining boards can now be tiled.

End of solution.

Exercise 8. Prove that any family of $> s!(k-1)^{s+1}$ (possibly equal) sets of size s has a sunflower of size k.

Solution: The sets in our collection may be equal, so let us consider how many sets we can have which are equal to each other.

- If any k sets $A_1, A_2, \ldots, A_k$ are equal to each other, they are a sunflower of size k. This is because $A_i \cap A_j = A_i = A_j$ for any $i \neq j$.

- If we don't have k sets in the entire collection which are equal to each other, we have to consider how many distinct *groups* of equal sets we have in our collection. Note that the size of each group is at most $k-1$. We have at least $s!(k-1)^{s+1}$ sets in total, so the number of distinct groups must be at least

$$\frac{s!(k-1)^{s+1}}{k-1} = s!(k-1)^s.$$

  If we form a collection using one representative set from each group, we have a collection of at least $s!(k-1)^s$ sets which, by the proof given in class, must contain a sunflower of size k. □

End of solution.

Exercise 9. Ramsey theorem can be equivalently stated in terms of colors. Let K_n be the graph on n nodes with an edge between any two nodes. Ramsey theorem states that for any integers $s \leq 2, t \leq 2$ there exists a number $R(s,t)$ such that if we color the edges of K_n, where $n \geq R(s,t)$ with two colors Red and Blue, there are either s nodes such that all edges between them are Red, or t nodes such that all edges between them are Blue.

The exercise asks you to prove an extension to three colors: for any integers $s \leq 2, t \leq 2, u \leq 2$ there exists a number $R(s,t,u)$ such that if we color the edges of K_n, where $n \geq R(s,t,u)$ with three colors Red, Blue, and Green, there are either s nodes such that all edges between them are Red, or t nodes such that all edges between them are Blue, or u nodes such that all edges between them are Green.

Solution: This is very similar to the proof given in class for $R(s,t)$. Let $Q(w)$ be the claim, "For all integers $s \geq 2, t \geq 2, u \geq 2, w = s+t+u, \exists R(s,t,u) \in \mathbb{Z}$ such that any graph on $n \geq R(s,t,u)$ vertices has s nodes such that all edges between them are Red, or t nodes such that all edges between them are Blue, or u nodes such that all edges between them are Green." We prove this by induction on w.

Base Case: $w = 6$

In this case, $s = t = u = 2$. We can pick $R(s,t,u) := 2$. This is true because in any complete graph of at least 2 nodes, the single edge in the graph must be colored either Red, Green, or Blue. Therefore, $Q(6)$ holds.

Inductive Step: We assume $w > 6$ and $Q(w-1)$ is true. We need to prove that $Q(w)$ is true. To do that, we need to handle several cases.

- If $s = t = 2$ then pick $R(s,t,u) := u$. Any graph K_n with $n = u$ edges either has at least at least one red edge, satisfying $s = 2$, or at least one blue edge, satisfying $t = 2$, or all green edges, satisfying u.

- If $s = u = 2$ or $t = u = 2$, we can reason similarly to the case $s = t = 2$ to show that $Q(w)$ is true.

- If $s = 2, t > 2, u > 2$, then we have two cases to consider:
  - If the graph contains a Red edge, then $s = 2$ is satisfied and $Q(w)$ is true.
  - If the graph does not contain any Red edges, then we pick $R(s, t, u) := R(t, u)$ from the two color version of this problem. Since that version has been proven true in class, we know that $Q(w)$ is satisfied here as well.
- If $s > 2, t = 2, u > 2$ or $s > 2, t > 2, u = 2$ we reason similarly to the case when $s = 2, t > 2, u > 2$ to show that $Q(w)$ is true.
- If $s > 2, t > 2, u > 2$ we pick $R(s, t, u) := R(s-1, t, u)+R(s, t-1, u)+R(s, t, u-1)+1$. Consider any graph with at least $R(s, t, u)$ nodes. Let x be the first node. Say x has exactly g red edges, h blue edges, and i green edges. The number of nodes in this graph is $g+h+i+1$, which we know is at least $R(s-1, t, u)+R(s, t-1, u)+R(s, t, u-1)+1$. This means that $g \geq R(s-1, t, u)$ or $h \geq R(s, t-1, u)$ or $i \geq R(s, t, u-1)$. We know show that in any of these cases $Q(w)$ is true.
  - If $g \geq R(s-1, t, u)$, we apply the inductive hypothesis to the graph of the neighbors of x. That is, g is at least $R(s-1, t, u)$; hence, either it has $s-1$ nodes all connected by red edges, in which case by adding x we get s nodes all connected by red edges, or t nodes all connected by blue edges, or u nodes all connected by green edges. In all of these cases, $Q(w)$ is true.
  - If $h \geq R(s, t-1, u)$ or $i \geq R(s, t, u-1)$, we reason similarly to show that $Q(w)$ is true.

This concludes our proof. Now go have a sandwich like a pro. □

End of solution.

Lecture 3

3.5 Summary

We continued reading "Think Like the Pros" from Increasing Subsequence till the end of Random Variables, Expectation, and Variance. We also discussed the solution to Buffon's Needle Problem using the linearity of expectation.

3.6 Buffon's Needle Experiment

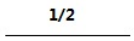
CAVEAT : We will use continuous probability space.
Experiment : Suppose you have a table with infinite number of parallel lines drawn on it, which are equally spaced with distance of 1 inch. Suppose you also have a needle, which is also 1 inch long. Throw the needle randomly on the table.
Question : What is the probability p that the needle intersects a line?

Solution: Let random variable X be the number of intersections. Since X can only take value 0 and 1, $E[X] = p$.

Now suppose you throw a needle with length 2 inches. Let random variable Y be the number of intersections produced by this needle of length 2. Consider dividing the needle into two halves with length 1. Let random variable X_1 be the number of intersections produced by the first half and X_2 be the number of intersections produced by the second half. Then $Y = X_1 + X_2$. Based on linearity of expectations: $E[Y] = E[X_1] + E[X_2] = p + p = 2p$

Similarly for the following needle which is not straight, we still have:
$E[$ # intersections of the needle with length $2] = 2p$

Also we have $E[$ #intersections of the needle with length $1/2] = p/2$



So we've seen that for a polygonal needle the expected number of intersections is a linear function $p * l$ of the length l of the needle.

Consider now a circular needle of radius $1/2$. If you drop the needle on the table, you will find that one of two things happens. (1) One line crosses the needle in two different points. (2) Two lines tangent to the needle. Therefore, the needle always intersects two lines and the expected number of intersections is 2. The length l of the needle = the circumference of the circle $= 2 * \pi * 1/2 = \pi$. Hence $2 = E[X] = p * l = p * \pi$. We get $p = 2/\pi$.

End of solution.

3.7 Exercises

Exercise 10. Prove $\forall k \geq 1$, $\forall n \geq k$, $(\frac{n}{k})^k \leq \binom{n}{k} \leq (\frac{en}{k})^k$.

Solution: Solution 1

First $(\frac{n}{k})^k \leq \binom{n}{k}$.

$$\begin{aligned}\binom{n}{k}/(\frac{n}{k})^k &= \Pi_{i=0}^{k-1} \frac{(n-i)k}{(k-i)n} \\ &= \Pi_{i=0}^{k-1} \frac{nk-ik}{nk-ni} \geq 1\end{aligned}$$

Second $\binom{n}{k} \leq (\frac{en}{k})^k$.

$$\begin{aligned}
\ln((\frac{en}{k})^k / \binom{n}{k}) =& k + k\ln n - k\ln k - \Sigma_{i=0}^{k-1}\ln(n-i) + \Sigma_{i=1}^{k}\ln i \\
\geq& k + k\ln n - k\ln k \\
& - (n\ln n - n - (n-k+1)\ln(n-k+1) + (n-k+1) + \ln n) \\
& + (k\ln k - k + 1) \quad \text{(summation of log to integral)} \\
=& (n-k+1)\ln(n-k+1) - (n-k+1)\ln n + k \\
=& (n-k+1)(\ln(n-k+1) - \ln n + \frac{k}{n-k+1}) \\
=& (n-k+1)(\ln(1 - \frac{k-1}{n}) + \frac{k/n}{1-\frac{k-1}{n}}) \\
=& (n-k+1)(-\frac{k-1}{n} - (\frac{k-1}{n})^2/2 - ... + \frac{k}{n}(1 + (\frac{k-1}{n}) + (\frac{k-1}{n})^2 + ...)) \\
& \text{(taylor expansion)} \\
=& (n-k+1)(\Sigma_{i=0}^{\infty}(\frac{k-1}{n})^i(\frac{k}{n} - \frac{k-1}{n(i+1)})) \geq 0
\end{aligned}$$

End of solution.

Solution: Solution 2

First we will prove $\forall k \geq 1, \forall n \geq k, \binom{n}{k} \geq (\frac{n}{k})^k$ by mathematical induction on k.

Base Case: $k = 1, \forall n \geq 1, \binom{n}{1} = n \geq (\frac{n}{1})^1$.

Inductive Hypothesis: Fix $k \geq 1$. Assume that

$$\forall n \geq k, \binom{n}{k} \geq (\frac{n}{k})^k.$$

Inductive Step: We want to prove that for fixed $k+1$,

$$\forall n \geq k+1, \binom{n}{k+1} \geq (\frac{n}{k+1})^{k+1}.$$

Since $n - 1 \geq k$, we have

$$\begin{aligned}
\binom{n}{k+1} &= \binom{n-1}{k}\frac{n}{k+1} \quad \text{(by definition)} \\
&\geq (\frac{n-1}{k})^k\frac{n}{k+1} \quad \text{(by the hypothesis)} \\
&\geq (\frac{n}{k+1})^k\frac{n}{k+1} \quad (\text{since } \frac{n-1}{k} \geq \frac{n}{k+1}) \\
&= (\frac{n}{k+1})^{k+1}.
\end{aligned}$$

Therefore we have proved that $\forall k \geq 1$, $\forall n \geq k$, $\binom{n}{k} \geq (\frac{n}{k})^k$.

Next we will prove that $\forall k \geq 1$, $\forall n \geq k$, $\binom{n}{k} \leq (\frac{en}{k})^k$ by mathematical induction on k.
Base Case: $k = 1$, $\forall n \geq 1$, $\binom{n}{1} = n \leq en$.
Inductive Hypothesis: Fix $k \geq 1$. Assume that

$$\forall n \geq k, \binom{n}{k} \leq (\frac{en}{k})^k.$$

Inductive Step: We want to prove that for fixed $k+1$,

$$\forall n \geq k+1, \binom{n}{k+1} \leq (\frac{en}{k+1})^{k+1}.$$

Since $n \geq k$, we have

$$\begin{aligned}
\binom{n}{k+1} &= \binom{n}{k}\frac{n-k}{k+1} \quad \text{(by definition)} \\
&\leq (\frac{en}{k})^k \frac{n-k}{k+1} \quad \text{(by the hypothesis)} \\
&= (\frac{en}{k})^k \frac{n-k}{k+1} \cdot \frac{en}{en} \cdot \frac{(k+1)^k}{(k+1)^k} \\
&= (\frac{en}{k+1})^{k+1} \cdot \frac{n-k}{n} \cdot \frac{(1+\frac{1}{k})^k}{e} \\
&\leq (\frac{en}{k+1})^{k+1} \cdot 1 \cdot 1 \quad \text{(since } (1+\frac{1}{k})^k \leq e\text{)} \\
&= (\frac{en}{k+1})^{k+1}.
\end{aligned}$$

Therefore we have proved that $\forall k \geq 1$, $\forall n \geq k$, $\binom{n}{k} \leq (\frac{en}{k})^k$.

End of solution.

Exercise 11. Prove $\forall n, a, b, c$ such that $n > abc$, any sequence of n (possibly equal) numbers has an increasing subsequence of length at least a, or a decreasing subsequence of length at least b, or a constant subsequence of length at least c.

Solution: Let $\{a_1, a_2, \cdots, a_n\}$ be a sequence of n numbers. For every i define $x_i \geq 1$ as the length of the longest increasing subsequence ending with a_i, $y_i \geq 1$ as the length of the longest decreasing subsequence starting with a_i, and $z_i \geq 1$ as the length of the longest constant subsequence ending with a_i.

Note that $(x_i, y_i, z_i) \neq (x_j, y_j, z_j)$, for $i < j$. Indeed, if $a_i < a_j$, then $x_j > x_i$; if $a_i > a_j$, then $y_i > y_j$; if $a_i = a_j$, then $z_j > z_i$.

So each triple (x_i, y_i, z_i) can only be assigned to one i. But there are less than n triples (x_i, y_i, z_i) with $x_i < a$, $y_i < b$, $z_i < c$. So some triple (x_i, y_i, z_i) with either $x_i \geq a$, or $y_i \geq b$, or $z_i \geq c$ must be assigned to some i.

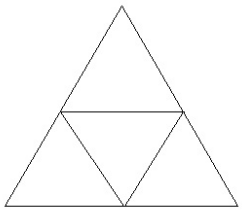
End of solution.

Exercise 12. Consider an equilateral triangle of side length 1.

Prove that no matter how you place 5 points inside it, two must be at distance $\leq 1/2$.

Now generalize this to prove that for every integer $k \geq 1$, if you place at least $4^k + 1$ points inside the triangle, two must be at distance $\leq 1/2^k$.

Solution: Partition the equilateral triangle of side length 1 into 4 small equilateral triangles of side length 1/2 as in the following figure.



According to pigeonhole principle, there must be two points in the same equilateral triangle of side length 1/2. The distance between these two points $\leq 1/2$.

If we partition each of these equilateral triangles of side length 1/2 into 4 equilateral triangles of side length 1/4, we get 4^2 equilateral triangles of side length $1/2^2$.

By repeating this process, we can get 4^k equilateral triangles of side length $1/2^k$. According to pigeonhole principle, there must be two points in the same equilateral triangle of side length $1/2^k$. So the distance between these two points $\leq 1/2^k$.

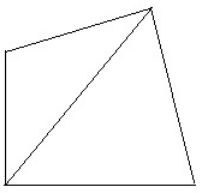
End of solution.

Exercise 13. Prove that in any graph with $2n$ nodes and at least $n^2 + 1$ edges, there is a triangle (a.k.a. a 3-clique).

Hint: Induction + Pigeonhole principle.

Solution: We prove this claim by mathematical induction. Let the set of edges in the graph be E, and the set of nodes in the graph be V.

Base Case: When $n = 2$, it is easy to verify that the only graph (up to isomorphism) with 4 nodes and 5 edges is:



Therefore, there is a triangle in the graph.

Inductive Step: Suppose the claim holds when $n = k$. That is, in any graph with $2k$ nodes and at least $k^2 + 1$ edges, there is a triangle.

When $n = k+1$, take any $(a, b) \in E$. Let $V_1 = \{a, b\}$, $V_2 = V - V_1$. $|V_2| = 2k$. There are two cases.

Case 1. $\exists y \in V_2$ such that $(a, y) \in E$, $(b, y) \in E$, so there is a triangle $\triangle aby$ in the graph.

Case 2. $\forall y \in V_2$, either $(a, y) \notin E$ or $(b, y) \notin E$. So there are at most $2k$ edges with one node in V_1 and the other node in V_2. So there are at least $(k+1)^2+1-1-2k = k^2+1$ edges with both of its nodes in V_2. According to the inductive assumption, there is a triangle in the subgraph induced by V_2.

End of solution.

Exercise 14. In this exercise you will show the existence of sets that are not regular without using the pumping lemma. Unlike the pumping lemma, the argument you will be using applies to a large number of definitions of "simple" sets. From a pedagogical point of view, this exercise is beneficial because lets you practice the pigeonhole principle and quantifiers and the distinction between finite and infinite.

Say that a set is t-regular if it is obtained as in Definition 10 where the integer k in that definition is constrained to be equal to t.

For $D \subseteq \{0,1\}^*$, say that a function $f : D \to \{0,1\}$ is (t-)regular if there exists a (t-)regular set S such that $\forall x \in D, f(x) = 1 \Leftrightarrow x \in S$.

(1) Prove that the number of t-regular sets is $2^{O(t \lg t)}$.

(2) Use (1) and the pigeonhole principle to prove that for sufficiently large t $\exists f_t : \{0,1\}^t \to \{0,1\}$ that is not t-regular.

(3) Use (2) to prove that there exists a set $S \subseteq \{0,1\}^*$ that is not regular.

Solution:

(1) Let $N(t)$ be the number of different sequences $(S_1, S_2, \cdots, S_t)$. For each $1 \leq i \leq t$, we consider the number of choices for S_i after choosing $S_1, S_2, \cdots, S_{i-1}$. Denote this number as $A(i)$. S_i has the following choices:

i) one of $\emptyset, \{0\}, \{1\}$ and ε, and there are 4 such choices;

ii) $\exists p, q < i$ such that $S_i = S_p \cup S_q$, and there are no more than $(i-1)^2$ such choices;

iii) $\exists p, q < i$ such that $S_i = S_p \circ S_q$, and there are no more than $(i-1)^2$ such choices;

iv) $\exists p < i, S_i = S_p^*$, and there are no more than $i-1$ such choices.

So $A(i) \leq 2(i-1)^2 + i - 1 + 4 = O(i^2)$. Therefore there exist constants $t_1 \geq 2$ and $c > 1$ such that $A(i) \leq ci^2$ for all $i \geq t_1$. When $t \geq t_1$,

$$N(t) = \prod_{i=1}^{t} A(i) = (\prod_{i=1}^{t_1-1} A(i))(\prod_{i=t_1}^{t} A(i))$$

Here $\prod_{i=1}^{t_1-1} A(i)$ is a constant, and we can denote it as M. Then

$$\begin{aligned} N(t) &= M \prod_{i=t_1}^{t} A(i) \le M \prod_{i=t_1}^{t} ci^2 \\ &\le M \prod_{i=1}^{t} ci^2 = Mc^t (t!)^2 \\ &\le Mc^t t^{2t} = 2^{2t \lg t + (\lg c)t + \lg M} \end{aligned}$$

Since $2t \lg t + (\lg c)t + \lg M = O(t \lg t)$, there exist constants $t_2 \ge 1$ and $d > 0$ such that $2t \lg t + (\lg c)t + \lg M \le dt \lg t$ for all $t \ge t_2$. Therefore $N(t) \le 2^{dt \lg t}$ for all $t \ge \max\{t_1, t_2\}$. Thus we have proved $N(t) = 2^{O(t \lg t)}$. The number of t-regular sets $\le N(t) = 2^{O(t \lg t)}$.

(2) Since the number of t-regular sets is $2^{O(t \lg t)}$, there exist positive constants c and t_0 such that the number of t-regular sets $\le 2^{ct \lg t}$ for all $t \ge t_0$.

Because $t \lg t = o(2^t)$, there exists a constant t_1, such that $2^t > ct \lg t$ for all $t \ge t_1$. Let $t_2 = \max\{t_0, t_1\}$. Then for $\forall t \ge t_2$, the number of t-regular sets $\le 2^{ctlgt} < 2^{2^t}$ = the number of different functions from $\{0,1\}^t$ to $\{0,1\}$.

Let $t \ge t_2$ be fixed. Assume $\forall f_t : \{0,1\}^t \to \{0,1\}$ is t-regular. According to pigeonhole principle, there are two different functions whose inverse images of $\{1\}$ are the same set. This implies these two functions are the same. We get a contraction. So the assumption is not correct. So $\forall t \ge t_2$, $\exists f_t : \{0,1\}^t \to \{0,1\}$ that is not t-regular.

(3) We don't know how to prove (3) using (2). Here is a direct proof of (3).

Let A_t be the set of all t-regular sets. Since every t-regular set is obtained by applying a finite number of operations on ε, 0, 1, $\emptyset$, the number of different t-regular sets is finite. So A_t is a finite set. Let A be the set of all regular sets. $A = \bigcup_{t \ge 1} A_t$ is a countable union of finite sets, so A is a countable set.

The elements in $\{0,1\}^*$ can be enumerated as ε, 0, 1, 00, 01, 10, 11, $\cdots$. Therefore $\{0,1\}^*$ is countably infinite. The set of all subsets of $\{0,1\}^*$ is a power set of a countably infinite set. So it is a uncountably infinite set. The cardinality of the set of subsets of $\{0,1\}^*$ is strictly larger than the cardinality of the set of all regular sets. So there exists a set $S \subseteq \{0,1\}^*$ that is not regular.

End of solution.

Exercise 15. Prove that if E_1 and E_2 are independent, then $\neg E_1$ and E_2 are also independent.

Solution: Since E_1 and E_2 are independent, $P[E_1 \cap E_2] = P[E_1]P[E_2]$. Therefore,

$$\begin{aligned} P[\neg E_1 \cap E_2] &= P[E_2] - P[E_1 \cap E_2] \\ &= P[E_2] - P[E_1]P[E_2] \\ &= (1 - P[E_1])P[E_2] \\ &= P[\neg E_1]P[E_2] \end{aligned}$$

So $\neg E_1$ and E_2 are also independent.

End of solution.

Exercise 16. Prove that $\exists \varepsilon > 0$ such that $\forall$ large enough n, we can find a subset $A \subseteq \{1, 2, \cdots, n\}$ such that
(1) $|A| > \varepsilon n^{\frac{1}{3}}$, and
(2) $\forall x, y, z \in A$, $x + y \neq z$.

Solution: Let $\varepsilon = 1$, $t = 3$. For each $n > t$, we have $n^2 > 8$, then $\frac{n}{2} > n^{\frac{1}{3}}$. We can take $A = \{x \in \{1, 2, \cdots, n\} \mid \text{x is odd}\}$.

Then $|A| \geq \frac{n}{2} > n^{\frac{1}{3}} = \varepsilon n^{\frac{1}{3}}$, and $\forall x, y, z \in A$, $x + y$ is even, z is odd, so $x + y \neq z$.

End of solution.

Exercise 17. Prove that $\exists \varepsilon > 0$ such that $\forall\ n \geq 2$, $\forall U$ containing n real numbers, we can find a subset $A \subseteq U$ satisfying
(1) $|A| > \varepsilon n^{\frac{1}{3}}$, and
(2) $\forall x, y, z \in A$, $x + y \neq z$.

Solution: Let $\varepsilon = 1/100$. When $n \geq 2$, we have

$$n^{1/3} - 10n^{1/3} = \frac{9}{10} n^{1/3} \geq \frac{9}{10} \cdot 2^{1/3} > 1.$$

So there is an integer $k \in [\frac{1}{10} n^{1/3}, n^{1/3})$. Let A be a set of size k, and A is generated by random sampling from U without replacement. That is, the first element is randomly selected from U, and the second element is randomly selected from the remaining $n-1$ elements of U, and so on. We know each element in U has the probability k/n of being selected into A, and any other element not in U has zero probability of being selected into A. Let random variable X_i denote the ith element selected into A. First we know

$$|A| = k \geq \frac{1}{10} n^{1/3} > 1/100 n^{1/3} = \varepsilon n^{1/3}.$$

Next we want to prove that

$$Pr[\forall x, y, z \in A, x + y \neq z] > 0.$$

Equivalently, we can prove

$$Pr[\exists x, y, z \in A, x + y = z] < 1.$$

In fact,

$$\begin{aligned}
&Pr[\exists x, y, z \in A, x + y = z] \\
=&Pr[\exists 1 \leq i \leq k, 1 \leq j \leq k : X_i + X_j \in A] \\
\leq&\sum_{i=1}^{k}\sum_{j=1}^{k} Pr[X_i + X_j \in A] \\
=&\sum_{i=1}^{k}\sum_{j=1}^{k}(Pr[X_i + X_j \in U]\frac{k}{n} + Pr[X_i + X_j \notin U]0) \\
\leq&\sum_{i=1}^{k}\sum_{j=1}^{k}\frac{k}{n} \\
=&\frac{k^3}{n} < \frac{n}{n} = 1.
\end{aligned}$$

End of solution.

Exercise 18. Perform Buffon's experiment. Describes the tools used and report the approximations you got after 10/20/30 throws.

Solution: We drew several equally spaced parallel lines on a large paper. The distance between two neighbouring lines is 4cm which is equal to the length of the needle we used. We threw the needle randomly onto the paper 30 times. We got 6 intersections in the first 10 throws, and the approximation of π is 10/3=3.33. We got 14 intersections in the first 20 throws, and the approximation of π is 20/7=2.86. We got 19 intersections in the first 30 throws, and the approximation of π is 60/19=3.16.

End of solution.

Exercise 19. (Extra Problem) Give a discrete proof of Buffon's Needle Problem. Specifically, for a parameter n, imagine throwing the needle uniformly at positions 0, 1/n, 2/n, ..., n-1/n with angles 0, $2\pi/n$, $4\pi/n$, ..., $2\pi(n-1)/n$. Show that $p = 2/\pi + o(1)$.

Lecture 4

4.8 Summary

In this lecture, we finished the topic in the *Think Like the Pros*, from **Random Variables, Expectation, Variance and all that** (which is section 9.5 in *Think Like the Pros*). We firstly introduced the concepts of *random variables, expectation and independent* etc, and went through some examples (e.g., toss coins) to strengthen our understanding.

Then we discussed the topic of *concentration of measure*, which includes *Markov's inequality*, *Chebychev's inequality*, *Chernoff bound*. We also introduced the definition of *Kullback-Leibler or Relative Entropy*, as well as some applications of above conclusions in the proofs and exercises, e.g., *Jenson's inequality*. The lecture ended at the discussion of examples of bad proofs.

Definition 10 (Chernoff bound)**.** Let $X_1, X_2, ..., X_n$ be independent 0/1 r.v. such that,

$$\text{If } Pr[X_i = 1] = p, \text{ then } \forall \delta \in [0, \infty)$$
$$Pr[\sum_{i=1}^{n} X_i \geq n(p + \delta)] \leq 2^{-nD(p+\delta||p)}.$$

Definition 11 (Kullback-Leibler divergence or Relative Entropy)**.** For discrete probability distributions $p(x), q(x)$, K-L divergence is $D(p||q) = \sum_x p(x) log_2 \frac{p(x)}{q(x)}$.

Fact: $D(p||q) \geq 0$

Proof.

$$\begin{aligned} -Dp||q &= \sum_x p(x) \lg \frac{q(x)}{p(x)} \\ &\leq \lg(\sum_x p(x) \frac{q(x)}{p(x)} \text{ (Jensen's inequality)} \\ &= \lg 1 = 0. \end{aligned}$$

□

4.9 Exercises

Exercise 20. In this exercise you will use concentration inequalities to prove anti-concentration inequalities, that is, the result that a certain random variable does deviate from its expectation. It is convenient to think of the experiment of tossing n $\{-1, 1\}$ coins (instead of $\{0, 1\}$ coins). Let S be the sum. Prove:

(1) $E[S] = 0$.

(2) $E[S^2] = n$.

(3) $E[S^4] = 3n^2 - 2n$. (This takes some patience.)

(4) The standard deviation of S^2 is at most $\sqrt{2} \cdot n$.

Now the idea is that if it was the case that S is often very close to its expectation 0, then also S^2 would be often very close to 0. But since S^2 has a much larger expectation (n) and a standard deviations about the expectation ($\sqrt{2} \cdot n$), this would violate a concentration inequality. Specifically:
(5) Use Cantelli's one-sided Chebychev's inequality to show this anti-concentration inequality:

$$Pr[|S| \geq \sqrt{n}/2] \quad \geq \quad \Omega(1).$$

Finally, answer this question:
(6) Could you have used Chebychev's inequality to infer (5) from (1)-(4)?

Note: Similar bounds can be obtained by approximations of the binomial coefficients such as $\binom{n}{n/2} = \Theta(2n/\sqrt{n})$ for n even. One benefit of the proof in $n/2$ this exercise is that it applies in other contexts as well. For example, if v is vector in R of length 1, with probability $\Omega(1)$ you have that $|\sum_i x_i v_i| \geq 1/2$.

Solution: Toss n coins. X_i is the -1/1 outcome of the i-th coin. $S := \sum_{i=1}^{n} X_i$.

$$x_i \quad = \quad \begin{cases} -1 & \text{if } x_i \text{ is head} \\ 1 & \text{if } x_i \text{ is tail} \end{cases}$$

(1) For X_i, by the definition of expectation, we have

$$\begin{aligned} E[X_i] &= Pr[X_i = -1] * (-1) + Pr[X_i = 1] * 1 = -1/2 + 1/2 = 0 \\ E[S] &= E[\sum_{i=1}^{n} X_i] \\ &= \sum_{i=1}^{n} E[X_i] \\ &= 0. \end{aligned}$$

(2)

$$\begin{aligned} E[S^2] &= E[\sum_i x_i^2 + 2\sum_{i<j} x_i x_j] \\ &= n + 2\binom{n}{2}(-1/2 + 1/2) \\ &= n. \end{aligned}$$

(3)

$$\begin{aligned} E[S^4] &= E[\sum_i x_i^4 + \binom{4}{2}\sum_{i<j} x_i^2 x_j^2 + 4\sum_{i<j} x_i^3 x_j + 4\sum_{i<j} x_i x_j^3 + 2\binom{4}{2}\sum_{i<j<k} x_i^2 x_j x_k \\ &\quad +2\binom{4}{2}\sum_{i<j<k} x_i x_j^2 x_k + 2\binom{4}{2}\sum_{i<j<k} x_i x_j x_k^2 + 4!\sum_{i<j<k<l} x_i x_j x_k x_l] \\ &= n + \binom{4}{2}\binom{n}{2} + 0 + 0 + 0 + 0 + 0 + 0 \\ &= 3n^2 - 2n. \end{aligned}$$

Here, I will explain $\binom{4}{2}\sum_{i<j} x_i^2 x_j^2$, we have $\binom{n}{2}$ ways to choose two different variables from $x_1, x_2, ..., x_n$, for each of these ways, we have $\binom{4}{2}$ ways to choose which two out of 4 is x_i. Other terms in the equation is of the same reasoning.

(4) By the definition of σ,

$$\begin{aligned} \sigma^2(S^2) &= E[S^4] - E^2[S^2] \\ &= 3n^2 - 2n - n^2 \\ &= 2n^2 - 2n \end{aligned}$$

.

The standard deviation is $\sigma = \sqrt{2n^2 - 2n} \leq \sqrt{2}n$.

(5)

$$Pr[|S| \geq \sqrt{n}/2] = Pr[S^2 \geq n/4]$$

Because $n/4 < \mu = n$, by Cantelli's one-sided Chebychev's inequality

$$Pr[X < \mu - k\sigma] \leq \frac{1}{1+k^2} \quad ,$$

$$\text{we have } Pr[S^2 \geq n/4] = 1 - Pr[S^2 < \mu - k\sigma] \geq 1 - \frac{1}{1+k^2}.$$

Let $n/4 = \mu - k\sigma$, where $\mu = n, \sigma \leq \sqrt{2}n$. So $k = \Theta(1)$, therefore $1 - \frac{1}{1+k^2} = \Omega(1)$.

(6) No. The Chebychev's inequality is limited to the measure that is $X > \mu + k\sigma$, where $k > 0$. In this case, $n/4 < \mu$, so $k < 0$.

End of solution.

Exercise 21. Recall Section 9.7. It is possible to show that $n = 6$ cannot be changed to $0.49n$, when sending strings of *bits*. However, suppose that instead of sending a string of n bits you send a string of n symbols in $[t] = \{1, 2, \ldots, t\}$. Suppose that the adversary changes at most $0.49n$ symbols. Show that $\exists t$ such that for sufficiently large n you can still send $2^{\Omega(n)}$ messages.

Solution: Similar to the proof in Section 9.7, we will prove that we can construct such a set C.

Let C consist of k strings of n symbols in $[t] = \{1, 2, \ldots, t\}$ independently at random. Then,

$$\begin{aligned} & Pr[\exists x, y \in C, x \neq y \text{ at distance } < 0.98n] \\ \leq\ & k^2 Pr[x, y \text{ at distance } < 0.98n] \qquad \text{(Union bound).} \end{aligned}$$

To bound $Pr[x, y \text{ at distance } < 0.98n]$, define z_i to be a $\{0, 1\}$ random variables as follows:

$$z_i \ = \ \begin{cases} 1 & \text{if } x_i = y_i \\ 0 & \text{otherwise.} \end{cases}$$

Note that $Pr[z_i = 1] = 1/t$. Now the probability $Pr[x, y \text{ at distance } < 0.98n]$ is the probability that the sum of the variable z_i is at least 0.02. By Chernoff bound,

$$\begin{aligned} & Pr[\sum_i z_i \geq 0.02n] \\ = \ & Pr[\sum_i z_i \geq n/t + \epsilon n] \\ \leq \ & 1/2^{\epsilon^2 n}. \end{aligned}$$

let $t = 100$, $\epsilon = 1/100$, we can set $k = 2^{\Omega(n)}$ and get the probability to be < 1, which guarantees the existence of such set.

End of solution.

Exercise 22. In this exercise you will see another example where probabilistic reasoning greatly simplifies counting. Suppose you toss n biased coins. The coins are independent, and each comes up heads with probability p. What is the probability that you get an even number of heads?

Hint: Think of the outcome of each toss as a number in $\{-1, 1\}$, and use $E[X \cdot Y] = E[X] \cdot E[Y]$ for independent random variables.

Solution: We define x_i to be a output of each toss as follows:

$$x_i \ = \ \begin{cases} -1 & \text{if } x_i \text{ is head} \\ 1 & \text{if } x_i \text{ is tail} \end{cases}$$

We define $X = \prod_{i=1}^{n} x_i$ as a random variable that

$$X = \begin{cases} 1 & \text{if the number of heads is even} \\ -1 & \text{if the number of heads is old} \end{cases}$$

Let X_k be the random variable that you get k heads when toss n biased coins. Obviously $X_k = 1$ when k is even. So the probability is

$$Pr[X_k = 1] = \binom{n}{k} p^k q^{n-k}.$$

Therefore, the probability that you get an even number of heads is:

$$\begin{aligned} Pr[\sum_{k}^{n} X_k = 1] &= \sum_{k \text{ is even}}^{n} \binom{n}{k} p^k q^{n-k} \\ &= \frac{1 + (1-2p)^n}{2}. \end{aligned}$$

End of solution.

The following is another solution for Exercise 22:

Solution: We define x_i to be a output of each toss as follows:

$$x_i = \begin{cases} -1 & \text{if } x_i \text{ is head} \\ 1 & \text{if } x_i \text{ is tail} \end{cases}$$

We define $X = \prod_{i=1}^{n} x_i$ as a random variable that

$$X = \begin{cases} 1 & \text{if the number of heads is even} \\ -1 & \text{if the number of heads is old} \end{cases}$$

We know that for the n coins, the number of heads is either odd or even. Therefore,

$$Pr[X = -1] + Pr[X = 1] = 1. \tag{1}$$

By the definition of expectation, we have

$$E[X] = Pr[X = -1] * (-1) + Pr[X = 1] * 1. \tag{2}$$

Since x_i and independent,

$$E[X] = \prod_{i=1}^{n} E[x_i] = (1-2p)^n. \tag{3}$$

By (1),(2),(3), We have

$$Pr[X=1]=\frac{1+(1-2p)^n}{2}.$$

.

End of solution.

Exercise 23. Use calculus, prove:

(1) $D(p||q)\geq 2(p-q)^2$.

(2) Come up with some $\epsilon>0$ such that,

$\forall p\leq\epsilon, D(2p||p)\geq\epsilon p$.

Solution: (1)

$$D(p||q)=p\dot{\ln}\frac{p}{q}+(1-p)\dot{\ln}\frac{1-p}{1-q}=p\dot{(}\ln p-\ln q)+(1-p)\dot{(}\ln(1-p)-\ln(1-q)).\quad(*)$$

If $p>q$, we have

$$\begin{aligned}(*)&=p\cdot\ln x|_q^p+(1-p)\cdot\ln(1-x)|_q^p\\&=\int_{x=q}^{p}\{\frac{p}{x}-\frac{1-p}{1-x}\}\mathrm{d}x\\&=\int_{x=q}^{p}\{\frac{p-x}{x(1-x)}\}\mathrm{d}x\\&\geq 4\int_{x=q}^{p}\{p-x\}\mathrm{d}x=2(p-q)^2.\text{(note that } x(1-x)\leq 1/4).\end{aligned}$$

Else if $p\leq q$, the proof is similar.

End of solution.

Solution: (2) Let $F(p)=D(2p||p)-\epsilon p=D(2p||p)=2p\cdot\ln\frac{2p}{p}+(1-2p)\cdot\ln\frac{1-2p}{1-p}-\epsilon p$. Since $F(p=0)=0$, so if we want to guarantee that $\forall p\leq\epsilon F(p)\geq 0$, we only need to make sure that $\forall p\in(0,\min\{1/2,\epsilon\})\frac{\mathrm{d}F}{\mathrm{d}p}\geq 0$.

$$\frac{\mathrm{d}F}{\mathrm{d}p}=-\frac{1}{1-p}-2\cdot\ln\frac{1-2p}{1-p}+2\ln 2-\epsilon.$$

Now let $x = 2 - \frac{1}{1-p}$, then $\max\{0, 2 - \frac{1}{1-\epsilon}\} \leq x < 1$ since $0 < p \leq 1/2$.

$$\begin{aligned}
&\frac{\mathrm{d}F}{\mathrm{d}p} = x - 2 - 2\ln x + 2\ln 2 - \epsilon \geq 0 \\
\iff &\frac{1}{2}x + (\ln 2 - 1 - \frac{1}{2}\epsilon) \geq \ln x \\
\iff &\text{when } x = 1,\ \frac{1}{2}x + (\ln 2 - 1 - \frac{1}{2}\epsilon) \geq \ln x \\
\iff &\ln 2 - \frac{1}{2} - \frac{1}{2}\epsilon \geq 0 \\
\iff &\epsilon \leq 2\ln 2 - 1\ (\approx 0.386).
\end{aligned}$$

So, pick any $\epsilon \in (0, 2\ln 2 - 1)$, we can fulfill the requirement.

End of solution.

Exercise 24. Prove Chernoff Bound.

Solution:

$$\begin{aligned}
Pr[\sum X_i \geq n(p+\delta)] &\leq Pr[e^{\lambda \sum X_i} \geq e^{\lambda n(p+\delta)}] \\
&\leq E[e^{\lambda(\sum X_i)}]/e^{\lambda(n(p+\delta))} (\text{Markov's inequality}) \\
(\text{Note that } E[e^{\lambda(\sum X_i)}] &= E[\prod_i e^{\lambda X_i}] = \prod_i E[e^{\lambda X_i}] = (e^\lambda p + (1-p))^n.) \\
&\leq \frac{(pe^\lambda + (1-p))^n}{e^{\lambda(n(p+\delta))}} \\
&= (\frac{(pe^\lambda + (1-p))}{e^{\lambda(p+\delta)}})^n.\ (*)
\end{aligned}$$

Let $q = p + \delta$, then (*) becomes $pe^{(1-q)\lambda} + (1-p)e^{-q\lambda}$.
See it as a function of λ, and take a *log* then compute its derivative:

$$\frac{d}{d\lambda} log(pe^{(1-q)\lambda} + (1-p)e^{-q\lambda}) = -q + \frac{pe^{(1-q)\lambda}}{pe^{(1-q)\lambda} + (1-p)e^{-q\lambda}}.$$

To get the maximum of (*), we let the derivative to be 0, so we have

$$q = \frac{pe^{(1-q)\lambda}}{pe^{(1-q)\lambda} + (1-p)e^{-q\lambda}} \implies \lambda = log\frac{q(1-p)}{p(1-q)}.$$

Notice that $q = p + \delta > p$, so $\lambda > 0$, which means that we get the maximum value of (*) at $log\frac{q(1-p)}{p(1-q)}$. Now we put this value of λ back into $log(pe^{(1-q)\lambda} + (1-p)e^{-q\lambda})$, and find the following fact

$$log(pe^{(1-q)\lambda} + (1-p)e^{-q\lambda}) = -qlog\frac{q}{p} + (1-q)log\frac{1-p}{1-q} = -D(q||p) = -D(p+\delta||p).$$

Now we get this result back into (*), and have $Pr[\sum X_i \geq n(p+\delta)] \leq e^{-nD(p+\delta||p)}$.

End of solution.

Exercise 25. Use Chernoff Bound to give a bound on the number of times need to perform Buffon's experiment to obtain α: $|\alpha - \pi| \leq 1/1000$ with probability ≥ 99.

Solution: Suppose we perform Buffon's experiment n times, and get m intersections, then we get $\alpha = \frac{2n}{m}$ as the approximation to π. If we want $|\alpha - \pi| < 1/1000$ with probability 99%, then equally we can make sure that $|\alpha - \pi| \geq 1/1000$ has probability 1%, which equals to $Pr[\alpha \geq \pi + 1/1000] + Pr[\alpha \leq \pi - 1/1000] \leq 1/100$.
Since $m = \sum_{i=1}^{n} X_i$, where X_i denotes the # of intersections we get during one Buffon's experiment ($X_i \in \{0, 1\}$), we have to make the following stand

$$Pr[\sum_{i=1}^{n} X_i \leq \frac{2n}{\pi + 1/1000}] + Pr[\sum_{i=1}^{n} X_i \geq \frac{2n}{\pi - 1/1000}] \leq 1/100. \ (*)$$

We know $\mu = \frac{2}{\pi}$. According to Chernoff's bound, we can get

$$Pr[\sum_{i=1}^{n} X_i \leq \frac{2n}{\pi + 1/1000}] \leq 2^{-n \cdot D(\frac{2}{\pi+1/1000}||\frac{2}{\pi})} = 0.6364^n.$$

$$\text{and } Pr[\sum_{i=1}^{n} X_i \geq \frac{2n}{\pi - 1/1000}] \leq 2^{-n \cdot D(\frac{2}{\pi-1/1000}||\frac{2}{\pi})} = 0.6368^n.$$

So, to make (*) stand, we let $0.6364^n + 0.6368^n \leq 1/100$, then we get $n \geq 12$. Therefore, we need to perform Buffon's experiment at least 12 times to get the required approximation.

End of solution.

Exercise 26. A Small-Intersection System (SIS) is a collection of m sets $S_1, S_2, \ldots, S_m$,

- $|S_i| \geq s$;
- $S_i \subseteq \{1, 2, \ldots, u\}$;
- $|S_i \cap S_j| \leq \alpha s$ for $\forall i \neq j$;

(1) give a construction with $\alpha = 0$ for some u;

(2) give a construction with any $u \geq c \cdot \lg(m^2)$, any $u \geq c \cdot s/\alpha$, for a constant c independent from m, s, u, α.
Hint for (2): Think $u = a \cdot s$, pick each set randomly with $Pr[x \in S] = 2/e$.

Solution: (1) If $\alpha = 0$, then $\forall i \neq j S_i \bigcap S_j = \emptyset$. So, if $u \geq m \cdot s$, then we can give a simple construction that $S_i = s \cdot (i-1), s \cdot (i-1) + 1, ..., s \cdot i - 1$ where $i \in 1, 2, ..., m$. But if $u < m \cdot s$, then we cannot construct a SIS of m sets, because we are short of elements.

End of solution.

Definition 12. Random variables $X_i, \ldots, X_n$ are k-wise independent if any k of them are independent.

Exercise 27. Consider the example we say after Chebyshev's inequalities.

- Argue that example only needs 2-wise independent.
- Give a bound $O(1/n^c)$ for $c > 1$, if coins are 4-wise independent.

Solution: (1) When you toss n 0/1 coins, suppose that these n coin flips are 2-wise independent. Let X_i be the output of the i^{th} coin-flippings,

$$X_i = \begin{cases} 1 & \text{if } i^{th} \text{ coin flip is head} \\ 0 & \text{if } i^{th} \text{ coin flip is tail} \end{cases}$$

Define random variable $X = \sum_{i=1}^{n} X_i$, by Chebyshev's inequality,

$$\begin{aligned} Pr[X \geq 3n/4] &= Pr[X \geq \mu + k\sigma] \\ &\leq Pr[(X-\mu)^2 \geq k^2\sigma^2] \\ &\leq 1/k^2. \end{aligned}$$

Because

$$\sigma^2(X) = \sum_{i=1}^{n} \sigma^2(X_i) + \sum_{i \neq j} Cov(X_i, X_j). \tag{4}$$

By definition, the covariance between any two 2-wise independent random variables is 0, i.e., $\sum_{i \neq j} Cov(X_i, X_j) = 0$ when X_i and Y_i are independent for any $i \neq j$, so in (4), $\sigma^2(X) = \sum_{i=1}^{n} \sigma^2(X_i)$.

Meanwhile, for the 0/1 random variables X_i, if $Pr[X_i = 1] = p$, then $\sigma^2(X_i) = p(1-p)$. Therefore $\sigma^2(X) = \sum_{i=1}^{n} \sigma^2(X_i) = np(1-p) = \Omega(n)$.

Let $k\sigma = 3n/4$, so $k^2\sigma^2 = \Omega(n^2)$, and then $k^2 = O(n)$. So probability of getting $\geq 3n/4$ heads is bounded by $O(1/n)$ when toss n 2-wise independent coins.

(2) Set $X = \sum_{i=1}^{n} X_i$ with mean μ and standard deviation σ, $Y := X - \mu$. Since

$Pr[Y \geq k\sigma] \leq Pr[Y^4 \leq k^4\sigma^4]$. By Markov's inequality, $Pr[Y \leq k\sigma] \leq E[Y^4]/k^4\sigma^4$, because

$$\begin{aligned} Pr[Y^4] &= E[(X-\mu)^4] \\ &= E[(\sum_{i=1}^{n}(X_i-\mu_i))^4] \\ &= \sum_{i,j,k,l} E[(X_i-\mu_i)(X_j-\mu_j)(X_k-\mu_k)(X_l-\mu_l)] \\ &= \sum_{i} E[(X_i-\mu_i)^4]. \qquad \text{(by 4-wise independent)} \end{aligned}$$

For the 0/1 random variables X_i, if $Pr[X_i = 1] = p$, then $E[(X_i-\mu_i)^2] = \sigma^2(X_i) = p(1-p)$. Therefore $\sum_i E[(X_i-\mu_i)^4] = \Omega(n^2)$.

Let $k\sigma = 3n/4$, so $k^4\sigma^4 = \Omega(n^4)$. So probability of getting $\geq 3n/4$ heads is bounded by $O(1/n^2)$ when toss n 4-wise independent coins, and hence the constant $c = 2$.

End of solution.

Lecture 5

5.10 Summary

In lecture we begin the unit on the computational power of various classes of languages, following the slide deck entitled 'slides-regular'. Specifically, we focused on regular languages. Formal definitions of Deterministic Finite Automata and Non-Deterministic Finite Automata were given along with examples of each. Regular languages where defined as languages that are accepted under some DFA. We saw how to convert a DFA into an NFA and proved the surprising result that DFA's and NFAs have the same computational power. We also went over proofs that regular languages are closed under the complement and union operations. □

5.11 Closure under the regular operations

Claim 13. The class of regular languages is closed under the complement operation.
Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $L(M) = A$,
Construct DFA $M' = (Q, \Sigma, \delta, q_0, F')$,
$F' := \overline{F}$,
$\forall$ string $w, w \in L(M') \Leftrightarrow w \notin L(M)$.

Proof. Let's prove ($\Rightarrow$)
Suppose $w \in L(M')$, so $w = w_1 w_2 \dots w_k$. $\exists$ sequence of $k+1$ states $r'_0, r'_1, \dots, r'_k$ where
1) $r'_0 = q_0$
2) $r'_{i+1} = \delta(r'_i, w_{i+1}) \; \forall \; 0 \leq i < k$
3) $r'_k \in F' = \overline{F}$
Since M and M' have same q_0, δ, any sequence of states $r_0, r_1, \dots, r_k$ showing that $w \in L(M)$ must be identical to $r'_0, r'_1, \dots, r'_k$. (by 1) and 2))
So in particular, $r'_k = r_k$ and so $w \notin L(M)$. (by 3)).
□

Claim 14. The class of regular languages is closed under the union operation.
We have two DFAs, M_A and M_B for regular languages A and B. N is the constructed NFA.
DFA $M_A = (Q_A, \Sigma, \delta_A, q_A, F_A)$,
DFA $M_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$,
NFA $N = (Q, \Sigma, \delta, q, F)$ where
$Q := \{q\} \cup Q_A \cup Q_B, F := F_A \cup F_B$,
$\delta(r, x) := \{\delta_A(r, x)\}$ if r in Q_A and $x \neq \epsilon$,
$\delta(r, x) := \{\delta_B(r, x)\}$ if r in Q_B and $x \neq \epsilon$,
$\delta(q, \epsilon) := \{q_A, q_B\}$.
We have $L(N) = A \cup B$

Proof. First, let's do $L(N) \subseteq A \cup B$.
Suppose $w \in L(M)$, so $w = w_1 w_2 \dots w_k$, $w_i \in \Sigma \cup \{\epsilon\}$. $\exists$ sequence of $k+1$ states $r_0, r_1, \dots r_k$

where
1) $r_0 = q$
2) $r_{i+1} \in \delta(r_i, w_{i+1})$
3) $r_k \in F$
By definition of N, $w_1 = \epsilon$ and $r_1 \in \{q_A, q_B\}$,
Without loss of generality, say $r_1 = q_A$, then $w = w_2 w_3 \dots w_k$ and we have a sequence of k states $r_1, r_2, \dots r_k$ which shows $w \in A$
Conversely, for $L(N) \supseteq A \cup B$.
Without loss of generality, suppose $w = w_1 w_2 \dots w_k$, $\exists$ a sequence $r_0 \dots r_k$ satisfies 1) - 3).
For M_A, $w = w_1 w_2 \dots w_k$, $w_0 = \epsilon$ and $r' = r_1 \dots r_k$, $r_0 = q$ which shows $w \in L(N)$, since $\delta(q, \epsilon) \in q_A$.

Claim 15. The class of regular languages is closed under the "$\circ$" operation.
Given two DFAs M_A and M_B which are:
DFA $M_A = \{Q_A, \Sigma, \delta_A, q_A, F_A\} : L(M_A) = A$,
DFA $M_B = \{Q_B, \Sigma, \delta_B, q_B, F_B\} : L(M_B) = B$,
Construct NFA $N = \{Q, \Sigma, \delta, q, F\}$ where:
$Q := Q_A \cup Q_B$, $q := q_A$, $F = F_B$,
$\delta(r, x) := \{\delta_A(r, x)\}$ if r in Q_A and $x \neq \epsilon$,
$\delta(r, \epsilon) = \{q_B\}$ if r in F_A,
$\delta(r, x) := \{\delta_B(r, x)\}$ if r in Q_B and $x \neq \epsilon$.

Proof. First, we prove $L(N) \subseteq A \circ B$.
Suppose we have $w \in L(N)$, so for $w = w_1 w_2 \dots w_l w_{l+1} \dots w_k \in \Sigma \cup \{\epsilon\}$, $\exists$ a sequence of $k+1$ states $r_0, r_1, \dots, r_k$ where
1) $r_0 = q_A$,
2) $r_{i+1} \in \delta(r_i, w_{i+1})$,
3) $r_k \in F_B$.
By the definition of N of NFA, $\exists 1 < l < k$ s.t. $\delta(r_{l-1}, w_l) = q_B$ where $w_l = \epsilon$ and $r_{l-1} \in F_A$.
Therefore, we can partition the string w into two non-overlapped parts, w' and w'', using w_l as the boundary, namely,

$$w' = w_1 w_2 \dots w_{l-1}, \quad w'' = w_{l+1} \dots w_k. \tag{5}$$

Recall the definition of the N, M_A, and M_B, we can find that w' (starting with state q_A and ending with state $r_{l-1} \in F_A$) can be accepted by M_A and w'' (starting with state q_B and ending with state $r_k \in F_B$) can be accepted by M_B. Naturally, we have $w' \in L(M_A)$ and $w'' \in L(M_B)$, which implies, $w = w'w'' \in A \circ B$.

Second, we prove $A \circ B \subseteq L(N)$.
Suppose we have two strings $w' = w_1 w_2 \dots w_{l-1}$ and $w'' = w_{l+1} \dots w_k$ and they are accepted

by M_A and M_B, respectively. Therefore, we know for these two strings, $\exists$ two state sequences r_1, r_2, r_{l-1} and $r_l \dots r_k$ that are accepted by M_A and M_B, respectively. By the definition of N and 1), 2), 3), we know that $\exists w_l = \epsilon$ s.t. $\delta(r_{l-1}, w_l) = r_l$ where $r_{l-1} \in F_A$ and $r_l = q_B$. Then the string $w = w'w''$ can be accepted by $L(N)$.
□

Claim 16. The class of regular languages is closed under the star operation. Given DFA $M_A = \{Q_A, \Sigma, \delta_A, q_A, F_A\} : L(M_A) = A$,
Construct NFA $N = \{Q, \Sigma, \delta, q, F\}$ where:
$Q := \{q\} \cap Q_A$, $F := \{q\} \cup F_A$,
$\delta(r, x) := \{\delta_A(r, x)\}$ if $r \in Q_A$ and $x \neq \epsilon$,
$\delta(r, \epsilon) := \{q_A\}$ if $r \in \{q\} \cup F_A$.

If $k = 1$, then we have $A^* = A$. $\Rightarrow A$ is closed under the star operation. Next, we focus on the situation when $k > 1$.
First, we prove $L(N) \subseteq A^*$.
Suppose $w \in L(N)$, so for $w = w_{11}w_{12} \dots w_{1l_1}w_{21} \dots w_{2l_2} \dots w_{k1} \dots w_{kl_k} \in \Sigma \cup \{\epsilon\}$, $\exists$ a sequence of $\sum_k l_k + 1$ states $r_0, r_1, \dots, r_{1l_1}, \dots, r_{kl_k}$ where
1) $r_0 = q$
2) $r_{i+1} \in \delta(r_i, w_{i+1})$
3) $r_{kl_k} \in F_A$
Then by the definition of $L(N)$, we know that there are $k-1$ transition functions s.t.

$$k-1 \begin{cases} \delta(r_{1l_1-1}, w_{1l_1}) = w_{21}, & \text{where } r_{1l_1-1} \in F_A, w_{1l_1} = \epsilon, w_{21} = q_A, \\ & \vdots \\ & \vdots \\ \delta(r_{k-1l_{k-1}-1}, w_{k-1l_{k-1}}) = w_{k1}, & \text{where } r_{k-1l_{k-1}-1} \in F_A, w_{k1} = \epsilon, w_{k1} = q_A. \end{cases}$$

Therefore, we can partition the string w into k strings $w_1, w_2, \dots, w_k$ using ϵ as the boundary, where

$$w_1 = w_{11} \dots w_{1l_1}, w_2 = w_{21} \dots w_{2l_2}, \dots, w_k = w_{k1} \dots w_{kl_k}.$$

Based on the definition of M_A, we know that each string in $w_1, w_2, \dots, w_k$ can be accepted by M_A. That is $w = w_1w_2 \dots w_k \in A^*$. Therefore, we have $L(N) \subseteq A^*$.

Second, we prove $L(N) \supseteq A^*$
Suppose we have k words that are all accepted by M_A, i.e., $w_1 \in A, \dots w_k \in A$ and $w_1w_2 \dots w_k \in A^*$. Therefore, for these k strings, $\exists k$ sequences of states that are all accepted by M_A. Recall the definition of NFA N and 1), 2), 3) we know that $\exists k-1$ transition

functions that can connect the states of these k strings s.t.

$$k-1\begin{cases} \delta(r_{1l_1-1}, w_{1l_1}) = w_{21}, & \text{where } r_{1l_1-1} \in F_A, w_{1l_1} = \epsilon, w_{21} = q_A, \\ & \vdots \\ & \vdots \\ \delta(r_{k-1l_{k-1}-1}, w_{k-1l_{k-1}}) = w_{k1}, & \text{where } r_{k-1l_{k-1}-1} \in F_A, w_{k1} = \epsilon, w_{k1} = q_A. \end{cases}$$

Therefore, these k strings plus $k-1$ transition functions can be accepted by $L(N)$. Since the connecting symbols $w_{1l_1}, w_{1l_2}, \ldots, w_{k-1l_{k-1}}$ are nothing but ϵ, we can conclude that $A^* \subseteq L(N)$.

5.12 Exercises

Exercise 28. Sipsor 1.31

Claim 17. Regular languages are closed under reversal
$\forall w, w \in L(M) \Leftrightarrow w^R \in L(M')$.

Solution: Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ which accepts language A
Construct NFA $M' = (Q', \Sigma, \delta', q_0', F')$ where,
$Q' := Q \cup \{q_0'\}$ $F' := q_0$
$\delta(r_B, x) := r_A$ if $r_A, r_B \in Q$ and $\delta(r_A, x) := r_B \in \delta$
$F \in \delta(q_0', \epsilon)$
We will show that if A is regular then $L(M') = A^R$.
Suppose $w \in L(M)$, so $w = w_1, w_2 \ldots w_k$ and $\exists$ a series of states $r_0, r_1, \ldots r_k$ where
1) $r_0 = q_0$
2) $r_i \in \delta(r_{i+1}, w_{i+1})$
3) $r_k \in F$
By definition $w^R = w_k, w_{k-1}, \ldots w_1$ and $F' = r_0$
Thus $\exists$ a sequence of states in $M' r_{k+1}', r_k', r_{k-1}', \ldots r_0'$
By definition of M, $r_{k+1}' = q_0'$. For every state in M we can do the reverse transiton in M' since $r_k' \in \delta(q_0', \epsilon)$ and $r_{k-i}' \in \delta(r_k', w_k')$.
Since $r_0' = q_0$ by 1) and $q_0 \in F'$ it must be that $w^R \in L(M')$ and Regular languages are closed under reversal.

End of solution.

Exercise 29. Sipsor 1.32
Let $\Sigma_3 = \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} \forall i, j, k \in \{0, 1\}\}$

Claim 18. If B = $\{q \in \Sigma_3^* |$ the bottom row of w is the sum of the top two rows $\}$
Then B is a regular language.

Solution: We will be working with B^R which is equivalent to B being regular by Problem 1.31
Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{q_A, q_O\}$
$\Sigma = \{a_{ijk} | a_{ijk} \in \Sigma_3\}$
$\delta(q_A, x) := q_A$ if $x \in \{a_{000}, a_{101}, a_{011}\}$
$\delta(q_A, x) := q_O$ if $x \in \{a_{110}\}$
$\delta(q_O, x) := q_O$ if $x \in \{a_{100}, a_{0110}, a_{111}\}$
$\delta(q_O, x) := q_A$ if $x \in \{a_{001}\}$
$q_0 := q_A$
$F := q_A$
Suppose $w \in L(M)$ and $w = w_1, w_2, ..., w_k$ $\exists$ a sequence of states $r_0, r_1, ..., r_k$. We will be working in B^r so we will be processing w from least significant bit to most significant. As we go along it is possible that the top two rows can overflow the bottom, but the converse is not true. If at any point the bottom row is greater than the top then the string is rejected. Thus at any state r_i then $w_i \forall 0 \leq i \leq k$ may be in two possible states q_A or q_O. If$r_i \in q_A$ then sequence the $w_1, w_2, ...w_i \in L(M)$. In otherwords, so far, the top two rows are equal to the bottom. If $r_i \in q_O$ it indicates that the top rows so far have overflowed the bottom row. The only way to escape q_O is if a later symbol is a_{001}, correcting the overflow. After processing w_k, $r_k \in q_A$ because there can not be underflow or overflow and since $q_A \in F$ it must be that $L(M)$ accepts B^r and thus accepts B by problem 1.31.
End of solution.

Exercise 30. Sipsor 1.33
Let

$$\Sigma_2 = \{\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}\}$$

Here, Σ_2 contains all columns of 0s and 1s of height two. A string of symbols in Σ_2 gives two rows of 0s and 1s. Consider each row to be a binary number and let

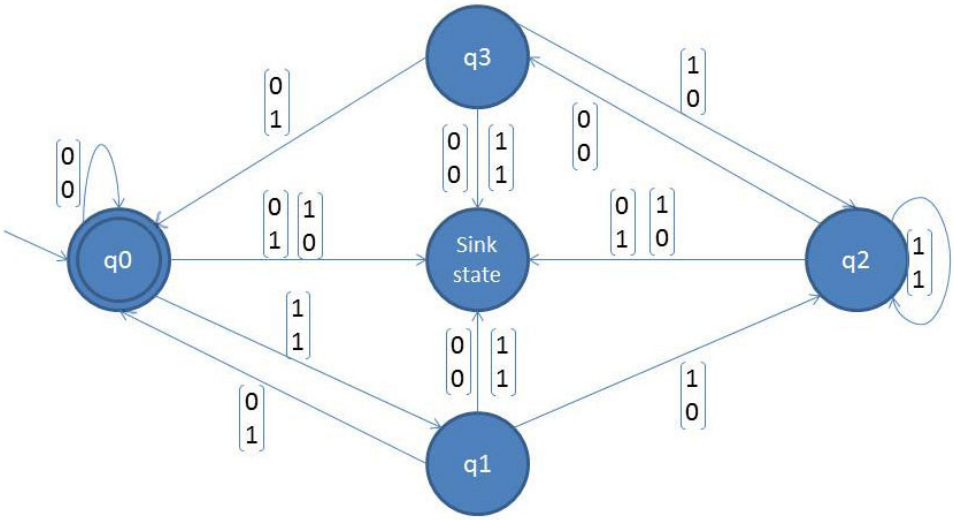
$$C = \{w \in \Sigma_2^* | \text{ the bottom row of } w \text{ is three times the top row}\}.$$

For example, $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} \in C$, but $\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} \notin C$. Show that C is regular.You may assume the result claimed in Problem 1.31.

Solution: According to Problem 1.31, if C is regular, so is C^R. So we can show C^R is regular.
Multiplying by 3 in binary is equivalent to adding the multiplicand with the result of shifting the multiplicand itself to the left by 1 bit. For a top row with n bits and a bottom row with n bits, denote them as $t = t_{n-1} \dots t_1 t_0$ and $b = b_{n-1} \dots b_1 b_0$ separately. Read w in

reverse order. If the bottom row is three times the top row, everytime we take a least significant bit t_i from top row and a bit b_i from bottom row, these conditions should be satisfied.

1) $b_i = t_i \bigoplus t_{i-1} \bigoplus c_i$,
2) $b_i = t_i$,
3) $c_{i+1} = (t_i \wedge t_{i-1}) \vee (t_i \vee t_{i-1}) \wedge c_i$ where
$a \bigoplus b = (a \wedge b) \vee (a \vee b)$, c_i is the carry-in bit and $t_{-1} = 0, c_0 = 0$.
Now we need to construct a DFA M with 4 states q_0, q_1, q_2, q_3 and a sink state.
q_0 indicates a state with carry-in bit $c_i = 0$ and $t_{i-1} = 0$,
q_1 indicates a state with carry-in bit $c_i = 0$ and $t_{i-1} = 1$,
q_2 indicates a state with carry-in bit $c_i = 1$ and $t_{i-1} = 1$,
q_3 indicates a state with carry-in bit $c_i = 1$ and $t_{i-1} = 0$.
Transition rules are shown on the graph.
$\forall w \in C^R$, 1) to 3) are satisfied, transition will stop at q_0. $\forall w \notin C^R$, 1) to 3) are not satisfied the transition will stop either at the sink state or q_1, q_2, q_3, so $C^R \in L(M)$. C^R is regular.
Based on Problem 1.31, C is also regular.



End of solution.

Exercise 31. Sipsor 1.34
Let Σ_2 be the same as in Problem 1.33. Consider each row to be a binary number and let

$$D = \{w \in \Sigma_2^* | \text{ the top row of } w \text{ is a larger number than the bottom row}\}.$$

For example, $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} \in D$, but $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} \notin D$. Show that D is regular.

Solution: The idea is that we start by comparing the most significant bit of two rows. The larger numbered row will have a bigger bit compared to the smaller numbered row. We continue the comparison until we can decide which row is larger or we run out of all bits. Construct DFA $M = (Q, \Sigma_2, \delta, q_0, F)$ where

$Q = \{q_0, q_1, q_2\}$,
state q_0 indicates the larger one between the top row and the bottom row has not been decided, state q_1 indicates the top row is the larger number and state q_2 indicates the bottom row is the larger number.
$\delta(q_0, \begin{bmatrix}0\\0\end{bmatrix}) := q_0$,
$\delta(q_0, \begin{bmatrix}1\\1\end{bmatrix}) := q_0$,
$\delta(q_0, \begin{bmatrix}1\\0\end{bmatrix}) := q_1$,
$\delta(q_0, \begin{bmatrix}0\\1\end{bmatrix}) := q_2$,
$F := q_1$.
DFA M can only accept all the $w \in D$, so D is regular.
End of solution.

Exercise 32. Sipsor 1.36
Let $B_n = \{a^k|$ where k is a multipe of $n\}$.
Show that for each $n \geq 1$, the language is regular.

Claim 19. B_n is a regular language $\forall n \geq 1$

Solution: We can construct DFA $M_n = (Q, \Sigma, \delta, q_0, F)$, such that $L(M_n) = B_n$
$Q = q_i | \forall 0 \leq i \leq n$
$\delta(q_i, a) := q_{i+i} \forall 0 \leq i \leq n$
$\delta(q_n, a) := q_0$
$F := q_n$
Suppose $w \in L(M_n)$ and $w = w_1, w_2 ... w_{n*i}$ where $i \in \mathbb{N}$
there is a series of states $r_0, r_1, ..., r_{n*1}, r(n*1)+1, ..., r_{n*i}$
$r_0 = q_0$ and there are n transitions such that $\delta(q_i, a) = q_{i+1}$ thus $r_n = q_n$. By similar reasoning $r_{n*j} = q_n \forall 0 \leq j \leq i$ and becuse $q_n \in F$ it must be that M_n accepts B_n
End of solution.

Exercise 33. Sipsor 1.37
Let $C_n = \{x | x$ is a binary number that is a multiple of n$\}$.
Show that for each $n \geq 1$, the language C_n is regular.

Solution: Construct a DFA $M = (Q, \{0,1\}, \delta, q_0, F)$
$Q := \{q_0, ...q_{n-1}\}$
$\delta(q_i, j) := q_{((i*2+j) \mod n)} \forall 0 \leq i \leq (n-1)$, $j \in \{0,1\}$
$F := \{q_0\}$
Begin by noting that

1) $(a+b) \mod n = (a \mod n) + (b \mod n)$
2) x is a multiple of $n \Leftrightarrow x \mod n = 0$
The general idea is to label each state by the current remainder. We then take each digit, starting with the most significant, and based on the digit and the current value for the remainder, calculate the new running remainder. States q_0 to q_{n-1} are the current remainder of r_i's state for $x_{k-1}x_{k-2}...x_i \mod n$ after the DFA has taken $(k-i)$ bits from x. If at the end of the process we are in state q_0 (the remainder is 0) then we know the full string was divisible by n.
Suppose $x \in L(M)$ where $x = x_{k-1}, x_{k-2}, ..., x_0$ thus $\exists$ a series of states $r_k, r_{k-1}, ...r_0$. We start by taking the most significant bit, x_{k-1}, and denoting the result of $x_{k-1} \mod n$ as r_{k-1} where $r_{k-1} = q_{x_{k-1} \mod n}$
The next digit of x is x_{k-2}, so now we calculate $x_{k-1}x_{k-2} \mod n$, which corresponds to state r_{k-2}. If x_{k-2} is 0 then $r_{k-2} = q_{r_{k-1}*2 \mod n}$. Otherwise, if x_{k_2} is 1 then $r_{k-2} = q_{r_{k-1}*2+1 \mod n}$, this preserves our invariant that the label of our current state is the remainder of the string processed so far. This process continues until we get r_0. If $r_0 = q_0$, then x is a multiple of n, otherwise x is not a multiple of n.

End of solution.

Exercise 34. Sipsor 1.41
For languages A and B, let the shuffle of A and B be the language

$$\{w | w = a_1 b_1 \dots a_k b_k, \text{ where } a_1 \dots a_k \in A \text{ and } b_1 \dots b_k \in B, \text{ each } a_i, b_i \in \Sigma^*\}.$$

Show that the class of regular languages is closed under perfect shuffle.

Solution: Given two DFAs M_A and M_B which are:
$M_A = \{Q_A, \Sigma, \delta_A, q_A, F_A\}$,
$M_B = \{Q_B, \Sigma, \delta_B, q_B, F_B\}$,
Construct NFA $N = \{Q, \Sigma, \delta, q, F\}$ where
$Q = \{q\} \cup Q_A \cup Q_B$, $F = \{q\} \cup F_B$,
$\delta(r, x) = \delta_A(r, x)$ if $r \in Q_A$ and $x \neq \epsilon$,
$\delta(r, x) = \delta_B(r, x)$ if $r \in Q_B$ and $x \neq \epsilon$,
$\delta(r, \epsilon) = q_B$ if $r \in F_A$,
$\delta(r, \epsilon) = q_A$ if $r \in F_B$.

If $k = 1$, the current problem degenerates to the problem of operation "$\circ$" in the class. And if $b_1 = \emptyset, b_2 = \emptyset, \dots, b_k = \emptyset$, the current problem degenerates to the problem of A^* in the class. Now we focus on the other situations when $k > 1$ and $\forall b_i \cup_i b_i \neq \emptyset$.

Suppose $w \in L(N)$, so for

$$w = \underbrace{w_{11}^{(a)} w_{12}^{(a)} \dots w_{1l_1^a}^{(a)}}_{a_1} \underbrace{w_{11}^{(b)} w_{12}^{(b)} \dots w_{1l_1^b}^{(b)}}_{b_1} \dots \underbrace{w_{k1}^{(a)} w_{k2}^{(a)} \dots w_{kl_k^a}^{(a)}}_{a_k} \underbrace{w_{k1}^{(b)} w_{k2}^{(b)} \dots w_{kl_k^b}^{(b)}}_{b_k} \in \Sigma \cup \{\epsilon\}$$

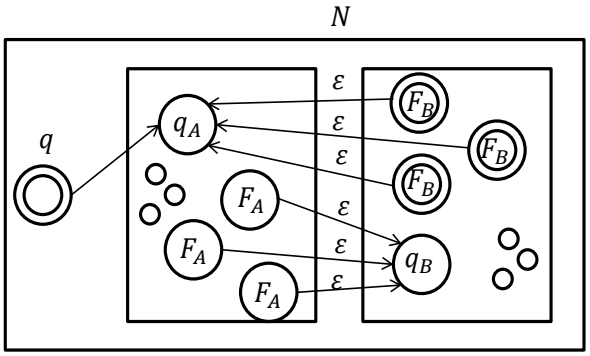


Figure 1: Illustration of NFA for the proof of problem 1.41.

, $\exists$ a sequence of $\sum_k l_k^a + \sum_k l_k^b + 1$ states $r_0^{(a)}, \ldots, r_{1l_1}^{(a)}, r_{11}^{(b)}, \ldots, r_{1l_1}^{(b)}, \ldots, r_{k1}^{(a)}, \ldots r_{kl_k}^{(a)}, r_{k1}^{(b)}, \ldots, r_{kl_k}^{(b)}$ where
1) $r_0 = q$,
2) $r_{i+1} \in \delta(r_i, w_{i+1})$,
3) $r_{kl_k} \in F_B$,

We could finish our proof following the same way in the proof of Claim 15 and 16 by using the above definition of NFA N. Or we can directly utilize Claim 15 and 16 to finish our proof. We pick up the second one since we can prove Shuffle(A, B) is closed in one pass based on Claim 15 and Claim 16.

Based on Claim 15, we know that $A \circ B$ is closed because both A and B are regular. In addition, based on Claim 16, we know that C^* where $C = A \circ B$ is closed because C is regular. It is clear that $C^* = \text{Shuffle}(A, B)$. So we have Shuffle$(A, B)$ is closed if both A and B are regular.

End of solution.

Lecture 6

After reviewing some of the exercises from past lectures, we continued with the lecture on regular languages and finite automata. We discussed a technique for proving whether a language is regular, and then moved on to regular expressions, an equivalent but more convenient formalism for describing regular languages.

6.13 The Myhill-Nerode Theorem

Recall that a language A is regular if there exists a DFA M such that $L(M) = A$. There is a theorem due to John Myhill and Anil Nerode that defines a necessary and sufficient condition for a language to be regular. Here we state it in terms of Boolean matrices.

Theorem 20. Given a language $A \subseteq \Sigma^*$, consider a matrix M_A with one row and column for every string in Σ^*, such that for any $x, y \in \Sigma^*$ the (x, y) entry is 1 iff $xy \in A$. Then A is regular iff M_A has finitely many distinct rows.

Proof. The proof of the "$\Rightarrow$" direction is left as an exercise. For the "$\Leftarrow$" direction, suppose M_A has finitely many distinct rows. Write $[x]$ for the rows equal to row x. We define a DFA $M = (Q, \Sigma, \delta, q_1, F)$ that recognizes A in the following way.

1. The states $Q = \{[x] \mid x \in \Sigma^*\}$ are the distinct rows.
2. Define δ so that for any $[x] \in Q$ and any $a \in \Sigma$, $\delta([x], a) = [xa]$.
3. The start state is $q_1 = [\epsilon]$.
4. The accept states $F = \{[x] \mid x \in A\}$ are the rows for strings in the language.

The proof that $L(M) = A$ is by straightforward induction. Intuitively, consider any string $x = x_1 x_2 \cdots x_k \in A$. The machine evaluates $\delta([\epsilon], x_1) = [x_1]$, $\delta([x_1], x_2) = [x_1 x_2]$, and so on until reaching $[x] \in F$. Thus if M_A has finitely many distinct rows, A is regular. $\square$

Exercise 35. Complete the proof of the above theorem.

Solution: For the "$\Rightarrow$" direction, suppose A is regular. Let M be a DFA recognizing A. For each state q_i, let $[q_i]$ be the set of strings x such that M ends at q_i on input x. Then for any $x, y \in [q_i]$ and any string z, it must be that xz and yz also end at the same state q_j. So if $x, y \in [q_i]$, then $xz, yz \in [q_j]$; either $xz, yz \in A$ and $q_j \in F$, or $xz, yz \notin A$ and $q_j \notin F$.

Therefore $x, y \in [q_i]$ implies $xz \in A$ iff $yz \in A$ for any z. But recall $[x] = [y]$ also means $xz \in A$ iff $yz \in A$ for any z, so we have that $x, y \in [q_i]$ implies $[x] = [y]$. Since every finite string maps to some $[q_i]$, and since no two strings that end up in a given $[q_i]$ map to more than one distinct row of M_A, the number of states in M must be no greater than the number of distinct rows in M_A. As A is regular and recognized by a finite automaton, M_A has finitely many distinct rows.

End of solution.

Exercise 36. Use this theorem to show that $A = \{0^n 1^n \mid n \geq 0\}$ is not regular.

Solution: Let $x = 0^i$ and $y = 0^j$, $i > 0$ and $i \neq j$, be two strings in $\{0,1\}^*$. If $[x] = [y]$, then for any z we must have $xz \in A$ iff $yz \in A$. But let $z = 1^i$. Then $xz \in A$ and $yz \notin A$, so $[x] \neq [y]$, *i.e.*, $[x]$ and $[y]$ are distinct rows in M_A. It follows that there are infinitely many distinct rows in M_A, because there are infinitely many strings such as x and y.

End of solution.

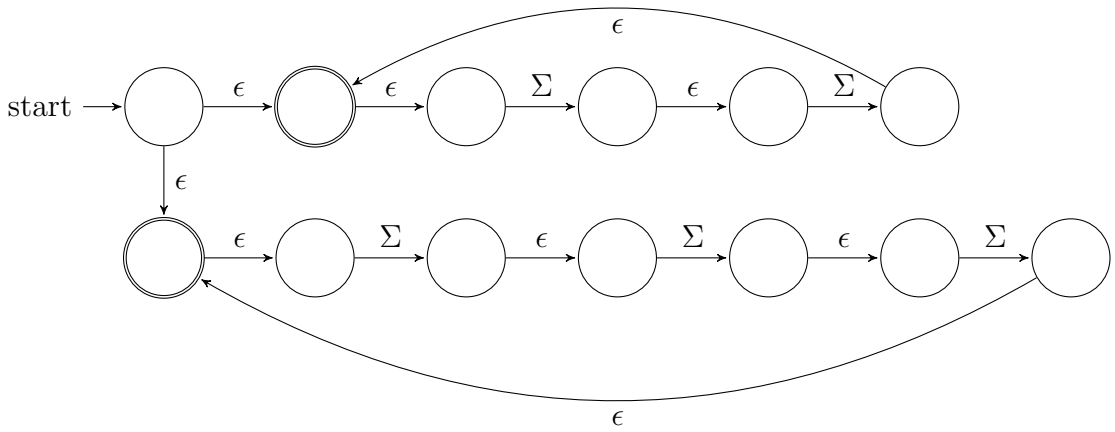
6.14 Regular Expressions

It is often tedious and complicated to describe a regular language with a finite automaton, either by writing out the formal definition or drawing a graphical representation. Regular expressions offer a more human-friendly alternative.

Regular expressions are built out of symbols from Σ and operations such as $*$, $\circ$, and $\cup$ that we have seen before. Indeed, they are equivalent to finite automata: we can convert a regular expression into an equivalent NFA and vice versa. The transformation from a regular expression to an NFA is simple in its base cases, and follows the closure constructions we saw earlier for the operations in the inductive cases.

Exercise 37. Build (using the process seen in class) an NFA for $(\Sigma\Sigma)^* \cup (\Sigma\Sigma\Sigma)^*$.

Solution: We begin with two-state machines for each Σ and end with the following:



End of solution.

The conversion from a finite automaton to a regular expression is more involved. Given a DFA, the first step is to turn it into a *generalized* NFA, or GNFA, that is like an NFA except for the transition function—the transitions are labeled with regular expressions describing strings instead of single symbols from the alphabet. The next step is to iteratively remove states from the GNFA, updating the transition labels, until only two states and one transition remain. The final label is the equivalent regular expression.

Exercise 38. A *small intersection system* (SIS) is a collection of m sets $S_1, S_2, \ldots, S_m$ such that $|S_i| \geq s$, $S_i \subseteq \{1, 2, \ldots, u\}$, and $|S_i \cap S_j| \leq \alpha s$ when $i \neq j$.

(1) Give a construction with $\alpha = 0$ for some u.

(2) Give a construction with $s \geq c \lg(m^2)$ and $u \geq cs/\alpha$, for a constant c independent of m, s, u, α. Hint: think of $u = as$, for a constant a, and pick each set randomly with $\Pr[x \in S_i] = 2/a$. Apply the Chernoff bound.

Solution: Note this is originally Exercise 26.

(1) Let $u = ms$ and $S_i = \{s(i-1) + j \mid 1 \leq j \leq s\}$. Then $|S_i \cap S_j| = 0$ when $i \neq j$ and $S_1, S_2, \ldots, S_m$ is an SIS. The numbers are simply distributed evenly across the m sets. For example, if $m = 3$ and $s = 2$, then $S_1 = \{1, 2\}$, $S_2 = \{3, 4\}$, and $S_3 = \{5, 6\}$.

(2)

End of solution.

Lecture 7

7.15 Summary

We continued going through slides-regular.pdf, beginning with the conversion of a DFA into a GNFA, and a GNFA into a RE. We then discussed the Pumping Lemma, which can be used to prove that some languages are not regular.

7.16 Converting DFAs into REs

Theorem 21. For any GNFA there is an equivalent Regular Expression.

Proof. By construction (and illustrated in the slides).

- If the GNFA contains exactly two states, then the label on the transition between them is the Regular Expression.
- If the GNFA contains more than two states, then iterate the following process until we are left with two:
    1. Select some interior state q_x (neither the start state nor the accept state), and remove it from the GNFA.
    2. Update the regular expressions for every pair of states q_i and q_j which were formerly connected through the removed state (that is, the path $q_i \to q_x \to q_j$ existed in the prior version of the GNFA) so that the same strings are recognized.

□

So a DFA, NFA, GNFA, or RE accept exactly the same regular languages.

What is the point of these conversions? They show that the set of regular languages is interesting: they have several characterizations. Also, they are used in practice all the time.

7.17 The Pumping Lemma

Which languages are not regular? We have already proved in the homework that the language $\{0^n 1^n : n > 0\}$ is not regular. The most powerful way to prove that a language is not regular is with the matrix proof, because it is exact. However, another commonly-used tool is the Pumping Lemma.

The main idea of the pumping lemma is that most DFAs contain loops, and those loops can be traversed any number of times (even zero). In fact, this is the only way that a DFA can accept a string which has more characters than the DFA has states: you must have gone through a loop. More interestingly, you didn't have to go through that loop, and you could have gone through it as many times as you want. (Expert tip: these loops correspond to the Kleene $*$ in the corresponding regular expression. This is because neither concatenations nor unions lead to loops.)

Let's get a bit more formal. The definition is on page 262 of slides-regular.pdf. It boils down to this: for any string w from a language L whose length $|w|$ is larger than the number of states in the DFA for L, you can segment w into three parts $w = xyz$ such that the middle part, y, could be replaced with any number of copies of itself and the resulting string must still be in L.

Here is the proof, which is not found in the slides.

Proof. Let p be the number of states in some DFA for language L, and let w be any string in L with length at least p. By the pigeonhole principle, during the computation of the DFA on the first p symbols of w some state $q*$ must repeat. That is, the path from start state q_0 to accepting state q_a which accepts w must look like this:

$$q_0 \to \ldots \to q* \to \ldots \to q* \to \ldots \to q_a.$$

Let's call y the label on the self-loop which starts and ends on $q*$, x the label leading from q_0 to $q*$, and z the label leading from $q*$ to q_a. The string xy^iz must be accepted for any $i \geq 0$, because if you can repeat the loop once you can repeat any number of times. □

This is not useful for proving that something is regular, but its contrapositive is useful to prove that it is not. We went through several examples of using the Pumping Lemma to prove that a language is not regular in class; these examples can be found in the slides.

7.18 Fun stuff about DFAs: 2DFAs

Why are DFAs so weak?

1. They have a finite number of states. This is a weakness, but even if you relaxed this you couldn't get to languages like $\{0^n1^n\}$.

2. You can only go in one direction: the transitions are one-way.

What happens if you allow two-way access to input? Let's define a 2DFA, which allows you to go backward and forward through the input. The main difference to a DFA is that each transition is labeled with a symbol and either L or R, indicating whether you read the symbol to the left or to the right of the current position in the input string in order to travel along that edge. We also put end markers, #, on either end of the string. That is, if the input is w we will call it $\#w\#$. The acceptance condition is still that you must be in an accept state when the input string is over.

Note that DFA closure under reversal is a special case of this: you can go to the end, and then come back.

Theorem 22. 2DFAs are equivalent to 1DFAs.

Proof. Rough sketch only:

Consider some boundary between two symbols in the input string. You might go back and forth across this boundary several times, and then eventually accept the string. The resulting

sequence of states is called a *crossing sequence*: a sequence of states that corresponds to crossing a boundary on a 2DFA.

If the 2DFA accepts w, how long can the crossing sequences be? It can be at most $O(\#$ of states), because otherwise the 2DFA would never stop and so would not accept w. That is, if you are ever at the same state a second time while in the same position in the input sequence, you will repeat that sequence forever and never accept the string. Since each crossing sequence is of finite length, you can construct a 1DFA whose states are the crossing sequences. In the transition function, you use:

$\delta(\text{crossing sequence}, a) = \{\text{compatible crossing sequences}\}$. □

7.19 Fun stuff about DFAs: PFAs

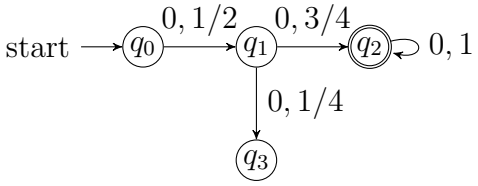
Next, we discussed Probabilistic Finite Automata (PFAs). First, we consider one-way PFAs, defined as follows. Given a state and symbol, you have a probability distribution over which state to go to next. You read a symbol and sample a random number from 0 to 1 and go to the corresponding state.

Definition 23. We say a 1PFA accepts or recognizes L if for all inputs x, if $x \in L$ then $Pr[\text{1PFA accepts } x] >= 99\%$, and if $x \notin L$ the probability $<= 1\%$.

Note that we define a 1PFA such that it must give a string a probability which is either $\geq 99\%$ or $\leq 1\%$ (or whatever you use for your definition). Note also that as an exercise, we show that the specific numbers 99% and 1% are not important. That is, you can pick any numbers for the thresholds as long as your 1PFA accepts strings which achieve higher probability.

You could think of a DFA as a 1PFA with probabilities 1 and 0.

Example:



$Pr[\text{accepts string } 00] = 1/2 + 1/2 * 3/4$

So, to summarize, a 2DFA is the same as a regular 1DFA. A 1PFA is also the same as a 1DFA. Surprisingly, if you combine the two you get extra power.

Theorem 24. There exists a 2PFA M such that $L(M) = \{a^n b^n, n \geq 0\}$.

Proof. The proof is a series of exercises. M does the following:

1. Check that the input is of the form $a * b*$. Let us say the input $w = a^n b^m$.
2. Reject if $0 < |n - m| < k$, for some k constant to be set later.

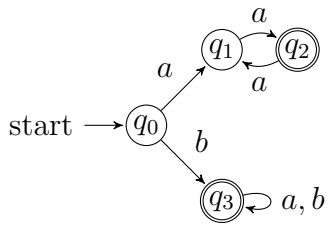
3. Scan input repeatedly, tossing a coin for each symbol. Success for a: always come up with 1, but some coin for b is 0. Success for b: All coins for b are 1, but some coin for a is 0. If $\exists L$ successes for a before any for b, or $\exists L$ successes for b before any for a, reject. Otherwise, accept. (L is a constant to be set depending on k.)

4. Proof completed in exercises.

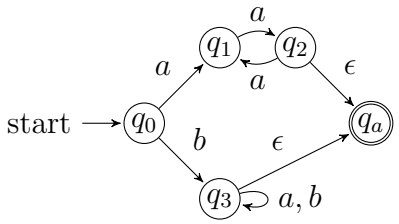
□

7.20 Exercises

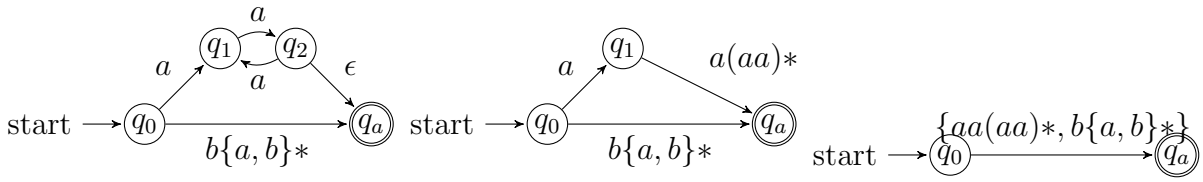
Exercise 39. Convert the following NFA into a RE:



Solution: The first step of this process is to convert the NFA into a GNFA. This is easy to do, in this case: we just have to create epsilon transitions from the two accept states to a single new accept state:



The next step is to reduce the GNFA one node at a time, until we are left with just q_0 and q_a. The regular expression for the single remaining transition will be our answer. Here is the process, taken step by step. We remove the nodes in descending numerical order (arbitrarily):



This gives us our answer: $\{aa(aa)*, b\{a,b\}*\}$.

End of solution.

Exercise 40. Think Like The Pros: Exercise 10. Prove that $\{x|x = wtw$ for some $w, t \in \{0,1\}(\{0,1\}*)\}$ is not regular (this is [S problem 1.46d]).

Solution:

Proof. By pumping lemma:

$$\forall p \geq 0, \exists w \in L, |w| \geq p, \forall x, y, z : w = xyz, |y| > 0, |xy| \leq p, \exists i \geq 0 : xy^i z \notin L$$

- Adversary moves p
- You select $w = 0^p 10^p$
- Adversary moves xyz
- You select $i = 0$

Since $|xy| \leq p$, y must consist of all zeros. When we remove it, the string contains fewer zeros to the left of the 1 than to the right. Therefore, $xz \notin L$ and this language is not regular. □

End of solution.

Exercise 41. Think Like The Pros: Exercise 11. Prove or disprove the claim: There exists a function $f(n) = \omega(n)$, with range the positive integers, such that $\{1^{f(n)}|n > 0$ is an integer$\}$ is regular.

Solution:

Claim 25. There is no such function $f(n)$.

Proof. To see this, one must consider the possible string lengths in a given regular language. The language specified above really consists of $f(n)$ 1s, for arbitrary positive integer n. However, the pumping lemma suggests that each string in a regular language is part of a family of similar strings whose lengths are a linear function of the number of times we have traversed some loop in that language. Since $f(n) = \omega(n)$, and since $f(n)$ returns only positive integers, no linear function returning positive integers can produce exactly the range of $f(n)$. A more formal proof follows, using the pumping lemma.

- Adversary moves p
- You select $w = 1^{f(n)}$, for any n such that $f(n) \geq p$
- Adversary moves xyz

- Note that $|w| = i|y| + |xz|$, for $i = 1$. That formula will give us the length of our selected string for any value of i we select. The language can only be regular if for any value of i, there is some value of n for which $f(n) = i|y| + |xz|$.

  Two facts combine to prevent this. Since $f(n) = \omega(n)$, for any constant c there is a corresponding constant n_0 such that for any $n \geq n_0$, $f(n) > cn$. The domain and range of $f(n)$ are integers, and at some point $f(n)$ will never come back down to any given value, so there is only a finite number of values of $f(n)$ which are smaller than any value. In particular, for sufficiently large n there are more possible values of $|w|$ than of $f(n)$. Therefore, there must be some i such that $|w| \notin \text{Range}(f)$; we pick one such i and win.

□

End of solution.

Exercise 42. Show that the set of languages accepted by a 1PFA does not change if we replace 99% and 1% with any a, b such that $1 > a > b > 0$.

Solution: if 1PFA accept a regular language L then it means: $\forall x \in L : Pr[$1PFA Accepts x $] \leq 1$ and $Pr[$1PFA Accepts x $] > 0$. so as long as we can find finite number of steps which start from initial state and follow a probability distribution according to observed alphabet in the input string toward the final state we can claim: $Pr[x \in L] > 0$ so by changing the threhold values a, b we only remove some strings from L and add it to another set with lower threshold values. but the overall strings accepted by the 1PFA will be the same.

End of solution.

Exercise 43. Show that 1PFAs accept exactly the regular languages. Some hints:

- Prove this using the theorem we showed last time. That is, given a 1PFA for L, show that the matrix M_L discussed last time has finitely many rows. This means that the language has a 1DFA.
- Row x of M_L will correspond to a probability distribution on states after reading x.
- You need to discretize the probabilities. That is, consider two probabilities equal if they differ by a tiny bit.

Solution: if we construct the M_L matrix for a given 1PFA, similar to the DFA approach, only with considering this fact that, for every string x which 1PFA ends at state q_i we have a set of different probailities for that row. so by discretizing the probabilities to bins with equal sizes, we only need to duplicate each row for the total number of created bins, representing the strings created with different probability values, and as we have finite number of rows for the equivalent DFA, then multiplying that with a constant factor bin_s will still remain as a finite number.

End of solution.

Exercise 44. Write a DFA to reject a string $w = a^n b^m$ if $0 < |n-m| < k$, for some constant k.

Solution:

End of solution.

Exercise 45. Complete the proof that a 2PFA can accept languages which a DFA can't. You just have to analyze why it works. That is, show that if the number of as is the same as the number of bs we will accept with high probability, and otherwise we will reject with high probability.

Solution: Since in the second step we have already filtered out the strings in which $0 < |n-m| < k$, we can assume that either $n = m$ or $|n-m| > k$.
if $n = m$ then both a s and b s have the same chance of success, therefore both the probability of L successes for a before any for b and the probability of L successes for b before any for a are $(\frac{1}{2})^L$. Since our machine rejects in case we get L consequtive successes, the probability of accepting a string with $n = m$ will be $1 - (\frac{1}{2})^L$. By selecting a big enough value for L we can get high probability of acceptance for $n = m$. For example by selecting $L = 4$ we get the probability of acceptance of 0.9375.
Now let's consider the case where $|n-m| > k$. The chance of success for as will be $(\frac{1}{2})^n$ since we need all n as to be get head in coin flips and the chance of success for bs will be $(\frac{1}{2})^m$. Therefore unlike the previous case as and bs have different chances of success and the probability of success for a compared to both as and bs will be:
$\frac{\frac{1}{2}^n}{\frac{1}{2}^m + \frac{1}{2}^n} = \frac{1}{\frac{1}{2}^{m-n} + 1}$
and the chance of rejecting the string because of L consequtive successes for a will be
$\left(\frac{1}{\frac{1}{2}^{m-n}+1}\right)^L$
and if $n > m$ then we can simplify to
$\left(\frac{2^k}{2^k+1}\right)^L$
We can see that selecting large values for k results in higher probability of rejecting strings with unequal number of as and bs.

End of solution.

Lecture 8

Exam 1 for PhD Core Theory of Computation, February 11, 2013

1. Prove by induction that the number of subsets of a set of size n is 2^n.

   Solution:

   Base case:

   $n = 1$. For set of size 1, its subsets are $\emptyset$ and itself. So there are $2 = 2^1$ subsets in total.

   Induction step:

   Suppose the number of subsets of a set of size $n-1$ is 2^{n-1}. For any set S of size n, pick any element s_i from the set. The subsets of S fall into two categories:

   (a) Subsets that don't contain s_i. There are the same number of these as the number of subsets of set $S - \{s_i\}$, which is 2^{n-1} by assumption.

   (b) Subsets that do contain s_i. Each of these is the union of $\{s_i\}$ and some subset of $S - \{S_i\}$. The number of such subsets is also 2^{n-1}.

   Based on the above, the number of subsets of set of size n is $2^{n-1} + 2^{n-1} = 2^n$.

   End of solution.

2. Recall that the degree of node in a graph is the number of its neighbors. (As in class, we consider simple graphs with no self-loops.) Prove that in any graph there are two nodes with the same degree.

   Solution:

   Proof. Consider a graph of n nodes. The degree of any node must be an integer in $\{0, 1, \ldots, n-1\}$. Suppose no two nodes have the same degree; then there must be some node in the graph with each degree in $0, 1, \ldots, n-1$. This is impossible because if any node has degree $n-1$, it means there are edges between this node and every other node in the graph, so no node can have degree 0. Therefore, there are two nodes with the same degree. □

   End of solution.

3. Let $C \subseteq \{0,1\}^n$ be a set of size $|C| \leq 2^{n/\lg n}$. Prove that, for all large enough n, there is a string $x \in \{0,1\}^n$ such that for every string $y \in C$, x has Hamming distance $\geq n/3$ from y. (Recall that the Hamming distance between x and y is the number of bit positions where they differ.)

   Solution:

Proof. Pick $y \in C$ randomly.

$$\begin{aligned}&\Pr[\exists x \in \{0,1\}^n, y \in C, x \neq y \text{ at distance} < n/3] \\ &\leq 2^{n/\lg n}\Pr[x, y \text{ at distance} < n/3] \qquad \text{(Union bound).}\end{aligned}$$

To bound $\Pr[x, y \text{ at distance} < n/3]$, rewrite x as $x_1x_2x_3 \ldots x_n$ and y as $y_1y_2y_3 \ldots y_n$, and define z_i to be a {0,1} random variable that is 1 if and only if $x_i = y_i$. Note $\Pr[z_i = 1] = 1/2$. Now the probability $\Pr[x, y \text{ at distance} < n/3]$ is the probability that the sum of the variables z_i is at least $2n/3$. By a Chernoff bound, this probability is at most $1/2^{n/36}$.

$$\Pr[\exists x \in \{0,1\}^n, y \in C, x \neq y \text{ at distance} < n/3] \leq \frac{2^{n/\lg n}}{2^{n/36}} = 2^{\frac{n(36-\lg n)}{36 \lg n}}$$

If we choose $n_0 = 2^{36}$, then for any $n > n_0$, $\Pr[\exists x \in C, x \neq y \text{ at distance} < n/3] < 1$, which guarantees that any two strings in C have hamming distance $\geq n/3$.
□

End of solution.

4. Recall that sets $A_1, A_2, \cdot, A_k$ are a sunflower of size k if $A_i \cap A_j$ is equal, for any $i \neq j$. Prove that any family of $> s!(k-1)^s$ distinct sets of size s contains a sunflower of size k.

Solution:

Proof. By induction on s:

Base case:

$s = 1$. We have $> (k-1)$ sets of size 1. Recall that they are distinct. Any k of these sets form a sunflower $A_1, \ldots, A_k$ with $A_i \cap A_j = \varnothing$, for any $i \neq j$.

Induction step:

Pick as many disjoint sets as you can. Call them $D_1, D_2, \ldots, D_t$.

If $t \geq k, D_1, D_2, \ldots, D_k$ is a sunflower of size k, with $D_i \cap D_j = \varnothing$ for any $i \neq j$.

If $t < k$, first observe that the total number of elements in the disjoint sets has the upper bound

$$\left|\bigcup_{i \leq t} D_i\right| \leq (k-1)s,$$

because each set has s elements, the disjoint sets all have different elements, and there are at most $k-1$ of these sets. Next, observe that any set in the family intersects some D_i. This is true because each set D_i intersects itself, and each set which is not some D_i

must intersect at least one of them or else it would be disjoint from all of them and be one of the D_i sets. Combining these observations, and using the pigeonhole principle, there must exist some element x that belongs to

$$\frac{\#\text{ total sets}}{\#\text{ elements in disjoint sets}} > \frac{s!(k-1)^s}{s(k-1)} = (s-1)!(k-1)^{s-1}$$

sets. Let $A_1, A_2, \ldots, A_u$ be these sets. By the inductive hypothesis, $A_1 - \{x\}, A_2 - \{x\}, \ldots, A_u - \{x\}$ contains a sunflower of size k. Since $A_1, A_2, \ldots, A_u$ all contain x their intersections are still equal, so they also contain a sunflower of size k. □

End of solution.

5. Prove that $\forall n, a, b$ such that $n > ab$, any sequence of n distinct numbers has an increasing subsequence of length at least a, or a decreasing subsequence of length at least b.

Solution:

Proof. By contradiction:
For every i define $x_i \geq 1$ as the length of the longest increasing subsequence ending with a_i, and $y_i \geq 1$ as the length of the longest decreasing subsequence starting with a_i. Note that $(x_i, y_i) \neq (x_j, y_j)$ for $i < j$. Indeed, if $a_i < a_j$ then $x_j > x_i$, while if $a_i > a_j$ when $y_i < y_j$. So each pair (x_i, y_i) can only be assigned to one i. But there are less than n pairs (x_i, y_i) with both $1 \leq x_i < a$ and $1 \leq y_i < b$. So some pair with one component larger must be assigned to some i. □

End of solution.

6. Prove that for all large enough n, for every set U containing n real numbers, we can find a subset $A \subset U$ satisfying

(1) $|A| > n^{\frac{1}{4}}$, and

(2) $\forall x, y, v, w \in A, x + y + v + w \neq 17$.

Solution:

Proof. By probabilistic method. Choose k items from U independently, at random. Call these items $a_1, a_2, \ldots, a_k$. Let A be the set of these items: $A = \{a_1, \ldots, a_k\}$. We need to show that the probability of violating either (1) or (2) is less than 1; this will show that some set A exists with the properties we want.

The probability of selecting any particular element from U is $\frac{1}{n}$. In order to show that A is sufficiently large we will show that none of our k elements have the same value, so that $|A|$ will equal k. We can talk about the specific size of k later. For any a_i and a_j where $i \neq j$, $Pr[a_i = a_j] = \frac{1}{n}$. By union bound, then, $Pr[a_i = a_j]$ for any $i \neq j$ is at most the number of events times the probability for each event, or $\leq \binom{k}{2}\frac{1}{n}$.

Now we need to find the probability of (2). Given any three elements x, y, and v, let us assume that the fourth number w such that the sum of all four equals 17 is a member of U. The probability of selecting this number is $\frac{1}{n}$. Given that, we need to find the probability that a bad fourth number will be selected for every set of three numbers in A. This works the same as above: we apply union bound and get $\leq \binom{k}{4}\frac{1}{n}$.

Now we can find the probability that $|A| < k$ or that any $x, y, v, w \in A$ sum to 17 with a final application of union bound. Our probability of failing is at most $\binom{k}{2}\frac{1}{n} + \binom{k}{4}\frac{1}{n}$. We need this to be less than 1, so we set $\binom{k}{2}\frac{1}{n} + \binom{k}{4}\frac{1}{n} < 1$, which is true when $\binom{k}{2} + \binom{k}{4} < n$. We can expand and simplify that, getting:

$$\begin{aligned} \binom{k}{2} + \binom{k}{4} &< n \\ \frac{k(k-1)}{2} + \frac{k(k-1)(k-2)(k-3)}{24} &< n \\ k^4 - 6k^3 + 23k^2 - 18k &< 24n \end{aligned}$$

If we take the fourth root of both sides of this equation, we find that we need a value smaller than k to be less than a value larger than $2n^{1/4}$. This leaves us with plenty of room for k itself to be larger than $n^{1/4}$. □

End of solution.

7. State the inequalities by Markov, Chebyshev, and Chernoff. Give definitions of all the quantities involved in these inequalities.

Solution:

Markov's inequality:

$$\Pr[X \geq k\mu] \leq \frac{1}{k}$$

Let X be a non-negative random variable with mean μ. Then for every $k > 0$, we have the above inequality.

Chebyshev's inequality:

$$\Pr[X \geq \mu + k\delta] \leq \frac{1}{k^2}$$

Let X be a non-negative random variable with mean μ and standard deviation σ. Then for every $k > 0$, we have the above inequality.

Chernoff's inequality: Let $X_1, X_2, ..., X_n$ be independent 0/1 r.v. such that,

$$\text{If } Pr[X_i = 1] = p, \text{ then } \forall \delta \in [0, \infty)$$
$$Pr[\sum_{i=1}^{n} X_i \geq n(p + \delta)] \leq 2^{-nD(p+\delta||p)}.$$

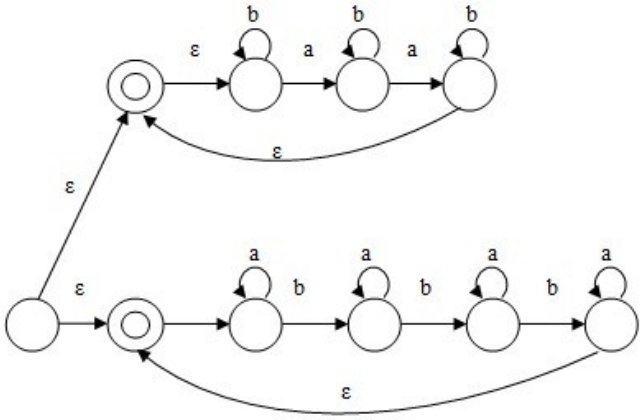
$D(p||q) = \sum_x p(x) log_2 \frac{p(x)}{q(x)}$, where $p(x), q(x)$ are discrete probability distributions.

End of solution.

8. Give NFA and RE for the language of strings over $\Sigma = a, b$ such that either the number of a is divisible by 2, or the number of b is divisible by 3.

Solution:

$M_{a \cup b} =$



Therefore the corresponding RE: $RE = (b^*ab^*ab^*)^* \cup (a^*ba^*ba^*ba^*)^*$.

End of solution.

Lecture 9

9.21 Summary

We solved the second part of Exercise 26. Then we started context-free slides and continued till context free pumping lemma.

9.22 Small-Intersection System (SIS)

Exercise 46. A Small-Intersection System (SIS) is a collection of m sets $S_1, S_2, \ldots, S_m \subseteq \{1, 2, \ldots, u\}$ where the followings hold,

(1) $|S_i| \geq s$;

(2) $|S_i \cap S_j| \leq \alpha s$ for $\forall i \neq j$;

Give a construction when u=es for e, sufficiantly large, depending on α and $s \geq c \cdot \lg(m)/\alpha$ for a constant c which is unversal.

Hint: Pick each set randomly with $Pr[x \in S] = \frac{2}{e}$.

Solution:

For a set S_i, let

$$X_j^i = \begin{cases} 1 & \text{if } j \in S_i \\ 0 & \text{otherwise} \end{cases}$$

Note $E[|S_i|] = E[\sum_{j=1}^{u} X_j^i] = \sum_{j=1}^{u} E[X_j^i] = u * \frac{2}{e} = e * s * \frac{2}{e} = 2s$
We want to show that with probability > 0 both (1) and (2) holds. To do so we can prove the following instead:
$Pr[Not(1) \text{ or } Not(2)] < 1$

Note $Pr[Not(1) \text{ or } Not(2)] \leq Pr[Not(1)] + Pr[Not(2)]$

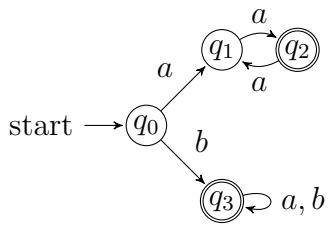
So we prove that each term is $< \frac{1}{2}$:
(1)
$Pr[\exists i : |S_i| < s] \leq$
$m.Pr[|S_i| < s] =$
$m.Pr[\sum_{j=1}^{u} X_j < s] =$
$m.Pr[\sum_{j=1}^{u} X_j < \frac{1}{e}.u] \leq m.2^{-D(\frac{1}{e}||\frac{2}{e})u}$
(2)
$Pr[\exists i \neq j || S_i \cap S_j| > \alpha s] \leq$
$m^2 Pr[|S_1 \cap S_2| > \alpha s]$
$m^2 Pr[|S_1 \cap S_2| > \alpha s] = m^2.Pr[|S_1 \cap S_2| \geq \frac{\alpha}{e}.u$

By chernoff bound, above probability is $\leq m^2.2^{-D(\frac{\alpha}{e}||(\frac{2}{e})^2)u}$
So we want each bound to $Pr < \frac{1}{2}$. With similar proofs to what has done in exercises:
$m^2.2^{-D(\frac{\alpha}{e}||(\frac{2}{e})^2)u} \leq m.2^{-\Omega(\frac{1}{e})u} = m.2^{-\Omega(s)}$
$m^2.2^{-D(\frac{\alpha}{e}||(\frac{2}{e})^2)u} \leq m^2.2^{-\Omega(\frac{\alpha}{e})u} = m^2.2^{-\Omega(\alpha)s} < \frac{1}{2}$

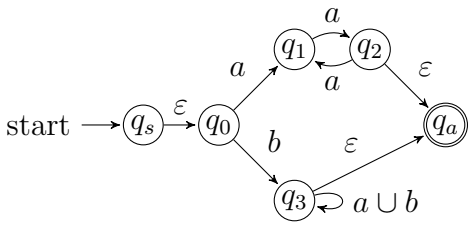
End of solution.

9.23 Some Previous Exercises

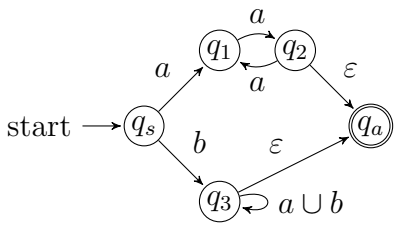
Exercise 47. Convert the following NFA into a RE:



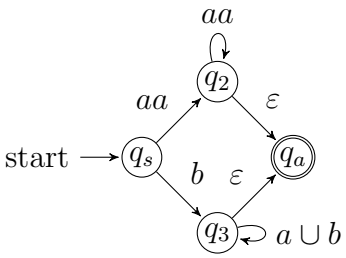
Solution: Add a new start state with an ε arrow to the old start state and a new accept state with an ε arrow from all old accept states.



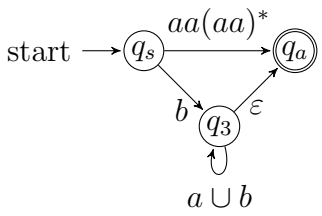
Remove q_0 and update labels.



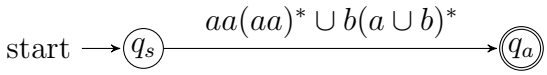
Remove q_1 and update labels.



Remove q_2 and update labels.



Remove q_3 and update labels.



We get the regular expression $aa(aa)^* \cup b(a \cup b)^*$.

End of solution.

Exercise 48. Think Like The Pros: Exercise 10. Prove that $\{x | x = wtw$ for some $w, t \in \{0,1\}(\{0,1\}*)\}$ is not regular (this is [S problem 1.46d]).

Solution:

Proof. By pumping lemma:

$$\forall p \geq 0, \exists w \in L, |w| \geq p, \forall x, y, z : w = xyz, |y| > 0, |xy| \leq p, \exists i \geq 0 : xy^i z \notin L$$

- Adversary moves p
- You select $w = 1^p 001^p 0$
- Adversary moves x, y, z
- You select $i = 2$

y only has 1 and $|y| > 0$, so $xyyz = 1^{p+|y|}001^p 0 \notin L$. □

End of solution.

Exercise 49. Think Like The Pros: Exercise 11. Prove or disprove the claim: There exists a function $f(n) = \omega(n)$, with range the positive integers, such that $\{1^{f(n)} | n > 0$ is an integer$\}$ is regular.

Solution:

Claim 26. There is no such function $f(n)$.

First we prove a lemma.

Lemma 27. If $f(n) = \omega(n)$, and $\left(n_i\right)_{i=1}^{\infty}$ is a subsequence of $\left(n\right)_{n=1}^{\infty}$, then $f(n_i) = \omega(i)$.

Proof of the above lemma.

$\forall c > 0$, $\exists n_0$ such that $f(n) > cn$ for all $n \geq n_0$. Pick i_0 such that $n_{i_0} \geq n_0$. So $f(n_i) > cn_i \geq ci$ for all $i \geq i_0$.

Proof of the claim.

- Adversary moves p
- You select $w = 1^{f(n)}$, for some n such that $f(n) \geq p$
- Adversary moves xyz
- We prove by contradiction that $\exists i, xy^iz \notin L$. Consider the set(not sequence) $\{xy^iz | i = 1, 2, \cdots\}$. If for each $i = 1, 2, \cdots$, $xy^iz \in L$, then there exists a subsequence $\left(n_i\right)_{i=1}^{\infty}$ of $(1, 2, \cdots)$ such that $\{xy^iz | i = 1, 2, \cdots\} = \{1^{f(n_i)} | i = 1, 2, \cdots\}$. The length of string in the increasing sequence $\left(xy^iz\right)_{i=1}^{\infty}$ is $\Theta(i)$. So $f(n_i) = O(i)$. This contradicts the lemma.

End of solution.

Exercise 50. Show that 1PFAs accept exactly the regular languages. Some hints:

- Prove this using the theorem we showed last time. That is, given a 1PFA for L, show that the matrix M_L discussed last time has finitely many rows. This means that the language has a 1DFA.
- Row x of M_L will correspond to a probability distribution on states after reading x.
- You need to discretize the probabilities. That is, consider two probabilities equal if they differ by a tiny bit.

Solution: Suppose the accept and reject thresholds are a and b, respectively. Suppose the PFA P has c non-halting states. Label them with $1, 2, \cdots, c$. Choose an ε small enough such that $c\varepsilon < a - b$. Partition $[0, 1]^c$ into finite number of small cells of dimension at most ε. So if $g, h \in [0, 1]^c$ are in the same cell, $\|g - h\|_\infty \leq \varepsilon$, where $\|\cdot\|_\infty$ is the maximum norm.

For a string x, let $p(x)$ be a vector in $[0, 1]^c$. The i-th entry of $p(x)$ is the probability that P starts in initial state and ends in state i after reading x. Let $q(x)$ be another vector in $[0, 1]^c$. The i-th entry of $q(x)$ is the probability that P starts in state i and ends in accept state after reading x. Then for two strings x and y, $p(x)q^T(y) =$Pr[P accepts xy].

Now suppose $p(x)$ and $p(x')$ are in the same cell, then $\|p(x) - p(x')\|_\infty \leq \varepsilon$. Then for any string y,

$$\begin{aligned}
|\Pr[P \text{ accepts } xy] - \Pr[P \text{ accepts } x'y]| &= |p(x)q^T(y) - p(x')q^T(y)| \\
&= |(p(x) - p(x'))q^T(y)| \\
&= |\sum_{i=1}^{c}(p_i(x) - p_i(x'))q_i(y)| \\
&\leq \sum_{i=1}^{c}|(p_i(x) - p_i(x'))q_i(y)| \\
&\leq \sum_{i=1}^{c}\varepsilon = c\varepsilon
\end{aligned}$$

So if P accepts xy, Pr $[P$ accepts $xy]> a$, Pr $[P$ accepts $x'y]> a - c\varepsilon > b$. So P also accepts $x'y$.

Therefore we have proved if x and x' are in the same cell, $[x] = [x']$. So the number of equivalence classes is no more than the number of cells, which is finite. According to Myhill-Nerode theorem, the language accepted by P is regular.

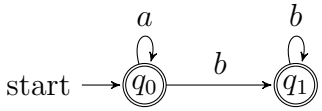
End of solution.

Exercise 51. There exists a 2PFA M such that $L(M) = \{a^n b^n, n \geq 0\}$.

Solution: The proof is a series of exercises. M does the following:

1. Check that the input is of the form $a * b*$. Let us say the input $w = a^n b^m$.

   The following DFA recognizes $\{a^n b^m | n, m \geq 0\}$.



2. Reject if $0 < |n - m| < k$, for some k constant to be set later.

   Define a DFA $(Q, \sum, \delta, q_0, F)$, where

   - $Q = \{0, 1, \cdots, k-1\} \times \{0, 1, \cdots, k-1\}$
   - $\sum = \{a, b\}$
   - $\delta((i,j), a) = (l, j)$, where $l \in \{0, 1, \cdots, k-1\}$ and $l \equiv i + 1 \bmod k$
     $\delta((i,j), b) = (i, p)$, where $p \in \{0, 1, \cdots, k-1\}$ and $p \equiv j + 1 \bmod k$
   - $q_0 = (0, 0)$
   - $F = \{(i,i) | i = 0, 1, \cdots, k-1\}$

Then this DFA accepts $w = a^n b^m$ if $n \equiv m \bmod k$. If $0 < |n - m| < k$, then this DFA rejects w.

3. Scan input repeatedly, tossing a coin for each symbol. Success for a: always come up with 1, but some coin for b is 0. Success for b: All coins for b are 1, but some coin for a is 0. If $\exists L$ successes for a before any for b, or $\exists L$ successes for b before any for a, reject. Otherwise, accept. (L is a constant to be set depending on k.)

   After step 1 and 2, there are two kinds of strings coming to step 3.

   - $w = a^n b^m$ with $n = m$
   - $w = a^n b^m$ with $|n - m| > k$ and $n \equiv m \bmod k$

   If $n = m$, Pr[2PFA accepts w]$\geq 1 - (\frac{1}{2})^{L-1}$. If $|n - m| > k$, Pr[2PFA accepts w]$\leq 1 - (1 - \frac{1}{2^k + 1})^L$.

   First we can choose L large enough to make $1 - (\frac{1}{2})^{L-1} \geq$ the accept threshold. Next we can choose k large enough to make $1 - (1 - \frac{1}{2^k + 1})^L \leq$ the reject threshold.

4. Compose the DFA in step 1, the DFA in step 2 and the 2PFA in step 3 together, we get a 2PFA which recognize $\{a^n b^n | n \geq 0\}$.

End of solution.

Exercise 52. Show that the set of languages accepted by a 1PFA does not change if we replace 99% and 1% with any a, b such that $1 > a > b > 0$.

Solution:

Let's generalize the problem a little bit for cleaner solution. Let's define two sets of PFAs; PFA_1 whose accepting probability is α and PFA_2 whose accepting probability is β, where $0 < \alpha, \beta < 1$. What we want to prove is that the two have same accepting power: for $\forall M_1 \in PFA_1$, $\exists M_2 \in PFA_2$ such that $L(M_1) = L(M_2)$.

Claim 28. For $\forall M_1 \in PFA_1$, create $M_2 \in PFA_2$ by duplicating M_1 but substituting $\forall$ transition probability p in M_1 to $p^{log_\alpha \beta}$. Then $L(M_1) = L(M_2)$.

Proof. For $\forall$ string w, if it arrives at final state with probability of $P(w)$ in M_1, it will arrive at final state with probability of $P(w)^{log_\alpha \beta}$ in M_2. Since $0 < \alpha, \beta < 1$, $f(x) = x^{log_\alpha \beta}$ is a monotonically increasing function. Because $0 \leqq P(w) \leqq 1$, $0 \leqq f(P(w)) \leqq 1$, and all converted probabilities in M_2 are valid probabilities. Because $f(x)$ is monotonically increasing and $f(\alpha) = \beta$, $L(M_1) = L(M_2)$. $\square$

End of solution.

9.24 Context Free Grammar Exercises

Exercise 53. Give context-free grammar for L={w | w has as many a as b}.

Solution:

Claim 29. Define a grammar G=(V, Σ, R, S) as follows, then L=L(G). Only definition of rules are presented below, because others are trivial.

$$S \to \epsilon|SS|aSb|bSa$$

Proof. $\Leftarrow$) All rules in G produces same amount of a and b. $\therefore \Leftarrow$.

$\Rightarrow$) Proof by induction on length of string, $|w|$.

Base case: $\epsilon \in L(G)$. trivial.

Induction step:

Induction hypothesis: For $\forall v \in L$, s.t. $|v| < |w|$, $v \in L(G)$.

Case 1. w is in the form axb, where $x \in \{a, b\}^*$. Since $w \in L$ and $|x| < |w|$, $x \in L$ and $x \in L(G)$ by induction hypothesis. Therefore, w can be produced by using rule $S \to aSb$ and $w \in L(G)$.

Case 2. w is in the form bxa, where $x \in \{a, b\}^*$. Similar to Case 1.

Case 3. w is in the form axa, where $x \in \{a, b\}^*$. Since w has same number of as and bs, when reading w from beginning, there must exists a point in the middle of w where there are same number of as and bs. Split w into two strings x and y at the point, e.g. $w = xy$. Then, $x, y \in L$, and by induction hypothsis, $x, y \in L(G)$. w can be created by using rule $S \to SS$ and therefore $w \in L(G)$.

Case 4. w is in the form bxb, where $x \in \{a, b\}^*$. Similar to Case 3. $\square$

End of solution.

Exercise 54. Give context-free grammar for L={w | w has twice as many a as b}.

Solution:

Claim 30. Define a grammar G=(V, Σ, R, S) as follows, then L=L(G). Only definition of rules are presented below, because others are trivial.

$$S \to \epsilon|SS|aSbSa|aSaSb|bSaSa$$

Proof. $\Leftarrow$) All rules in G produces twice the amount of a than b. $\therefore \Leftarrow$.

$\Rightarrow$) Proof by induction on length of string, $|w|$.

Base case: $\epsilon \in L(G)$. trivial.

Induction step: Induction hypothesis: For $\forall v \in L$, s.t. $|v| < |w|$, $v \in L(G)$.

Case 1. w is in the form bxb, where $x \in \{a, b\}^*$.

Since w has twice the amount of a than b, when reading w from the beginning, there must exists a point in the middle of w where the last character read is a and the number of a is twice that of number of b [count(a)=2 $\times$ count(b)]. Split w into two strings x and y at the point ($w = xy$). Then, $x, y \in L$, and by induction hypothsis, $x, y \in L(G)$. w can be created by using rule $S \to SS$ and therefore $w \in L(G)$.

Case 2. w is in the form axa, where $x \in \{a, b\}^*$.

Similar to Case 1, when reading w from the beginning, there must exist a point in the middle of w where the last character read is b and count(a)=2*count(b) or count(a)=2*count(b)-1.

If count(a)=2 $\times$ count(b), similar to Case 1., we can split w into two strings x and y at the point ($w = xy$), where $x, y \in L$ and by induction hypothsis, $x, y \in L(G)$. w can be created by using rule $S \to SS$ and therefore $w \in L(G)$.

If count(a)=2 $\times$ count(b)-1, we can split w as follows: ($w = axbya$). Then, $x, y \in L$, and by induction hypothsis, $x, y \in L(G)$. w can be created by using rule $S \to aSbSa$ and therefore $w \in L(G)$.

Case 3. w is in the form axb, where $x \in \{a, b\}^*$.

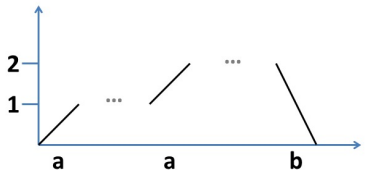


Figure 2: Change of count of a and b when a=1 and b=-2.

Similar to Case 1, as in the graph above, there must exist a point in the middle of w where the last character read is a and count(a)=2*count(b)+2 and we can split w as follows: ($w = axayb$), where $x, y \in L$ and by induction hypothsis, $x, y \in L(G)$. Then, w can be created by using rule $S \to aSaSb$ and therefore $w \in L(G)$.

Case 4. w is in the form bxa, where $x \in \{a, b\}^*$. Similar to Case 3, we can split w as follows: ($w = bxaya$) and $w \in L(G)$. □

End of solution.

Exercise 55. Give context-free grammar for L={w | w has as many a as b and each prefix of w has at least as many a as b }.

Solution:

Claim 31. Define a grammar G=(V, Σ, R, S) as follows, then L=L(G). Only definition of rules are presented below, because others are trivial.

$$S \to \epsilon | SS | aSb$$

Proof. $\Leftarrow$) Prefix of any rule in G has at least as many a as b. $\therefore\Leftarrow$.

$\Rightarrow$) Proof by induction on length of string, $|w|$. **Base case:** $\epsilon \in L(G)$. trivial.
Induction step: Induction hypothesis: For $\forall v \in L$, s.t. $|v| < |w|$, $v \in L(G)$.

Case 1. If there is a point in w, where count(a)=count(b).

Split w into two strings x and y at the point, e.g. $w = xy$. Then, $x, y \in L$, and by induction hypothsis, $x, y \in L(G)$. w can be created by using rule $S \to SS$ and therefore $w \in L(G)$.

Case 2. If there is no point in w, where count(a)=count(b).

Then count(a) > count(b) at all points in w except the end, and w must start with a and end with b: $w = axb$ where $x \in L$ from the assumption of the Case 2. By induction hypothsis $x \in L(G)$, and w can be created by using rule $S \to aSb$ and therefore $w \in L(G)$. □

End of solution.

Exercise 56. Give context-free grammar for L=$\{a^m b^n c^p d^q | m + n = p + q\}$.

Solution:

Claim 32. Define a grammar G=(V, Σ, R, S) as follows, then L=L(G). Only definition of rules are presented below, because others are trivial.

$$
\begin{aligned}
S &\to aSd|P|Q|R \\
P &\to aPc|R \\
Q &\to bQd|R \\
R &\to \epsilon|bRc
\end{aligned}
$$

Proof. $\Leftarrow$) All the strings generated by the rule is in the form of $a^m b^n c^p d^q$ and all the rules in G preserves the equation m+n=p+q. $\therefore\Leftarrow$.

$\Rightarrow$) For $\forall w \in L$, w can be produced by G as follows.

(1) Apply S → aSd for min(m,q) times. Then,

   (a) If $m = q$, apply S → R.

   (b) If $m > q$, apply S → P, apply P → aPc for (m-q) times, and apply P → R.

   (c) If $m < q$, apply S → Q, apply Q → bQd for (q-m) times, and apply Q → R.

(2) Apply R → bRc for min(n,p) times. Then, apply R → ϵ.

□

End of solution.

Exercise 57. Prove L=$\{0^n1^n0^n1^n | n \geqq 0\}$ is not context-free.

Solution:

Proof. by Context-Free Grammar Pumping Lemma.

For $\forall p \geqq 0$, let $w = 0^p1^p0^p1^p$. However w is divided into $uvxyz$, because $|vxy| \leqq p$, vxy cannot cover all 4 parts of alternating 0s and 1s of the string. Given that $|vy| > 0$, setting $i = 0$ will make the resulting string not to be in L. $\therefore$ L is not a context-free language.

□

End of solution.

Lecture 10

10.25 Summary

In this lecture, we mainly discuss the topic of Turing machine, decidable and undecidable problems.

10.26 Exercises

Exercise 58. Pick your favorite language,

(1) write your decider D for $\{\omega \in \Sigma^* | \omega \text{has even number of a}\}$,

(2) run D on itself and tell us the output,

(3) write a program that on input a program M, outputs a program M', where

$$M'(\omega) = \begin{cases} 0 & \text{if } \omega = \text{"hello"}, \\ M(\omega) & \text{otherwise} \end{cases}$$

(4) run it on D and output answer.

Solution: All of the following programs are written in `JAVA`.

Note: we include the source code in the folder `2013-02-20-g4-source-code`. There are three files:

(1) `Decider.java` is the decider D.

(2) `Program.java` is the *program* which operates on a input program M.

(3) `MainFunc.java` is the program used to test the above two programs.

You can use `javac` command to compile them and `java` command to execute them.

(1) Listing 1 is the decider D:

Listing 1: Decider D

```
1  public class Decider {
2          /**
3           * Decider D for L={w|w has even number of a}
4           *
5           * @param w
6           *             a string
7           * @return int: 1: even number; -1: odd number
8           */
9          public static int decider(String w) {
10                 int count = 0;
```

```
                for (int i = 0; i < w.length(); i++) {
                        if (w.charAt(i) == a) {
                                count++;
                        }
                }
                if (count % 2 == 0) {
                        return 1;
                } else {
                        return -1;
                }
        }
}
```

(2) Run D on itself, it returns 1, which means there are even number of a. Actually, there are four a's. Noting that we count the line `public class Decider`.

(3)

Listing 2: Program M

```
/**
 * @param M
 *              : the file name of the program
 * @param progName
 *              : program name
 */
public static void program(String M, String progName) {
try {
  File file = new File(M);
  if (!file.exists()) {
  System.out.println("File " + M + "does not exist.");
} else {
  File outfile = File.createTempFile(M, ".temp");

  FileInputStream fis = new FileInputStream(file);
  BufferedReader in = new BufferedReader(new InputStreamReader(
                fis));

  FileOutputStream fos = new FileOutputStream(outfile);
  PrintWriter out = new PrintWriter(fos);

  String line;

  Pattern pattern = Pattern.compile(progName);
  Matcher matcher;
  while ((line = in.readLine()) != null) {
  matcher = pattern.matcher(line);
```

```
28    if (matcher.find()) {
29          Pattern p = Pattern.compile("String␣w");
30          Matcher m = p.matcher(line);
31          if (!m.find()) {
32         line = line.substring(0,line.indexOf("(") + 1)
33                     + "String␣w,␣"
34                     + line.substring(line.indexOf("(") + 1);
35          }
36          out.println(line);
37          out.println("\t\t" + "if(w.equals(\"hello\")){");
38          out.println("\t\t\t" + "return␣0;");
39          out.println("\t\t" + "}");
40    } else {
41          out.println(line);
42    }
43    }
44    out.flush();
45    out.close();
46    in.close();
47
48    file.delete();
49    outfile.renameTo(file);
50   }
51   } catch (IOException e1) {
52          e1.printStackTrace();
53   }
54  }
```

(4) Run program on D, the output is shown in Listing 3. Line $10-12$ are the new added code:

Listing 3: The output after running program on D

```
1  public class Decider {
2          /**
3           * Decider D for L={w|w has even number of a}
4           *
5           * @param w
6           *             a string
7           * @return int: 1: even number; -1: odd number
8           */
9          public static int decider(String w) {
10                 if(w.equals("hello")){
11                         return 0;
12                 }
13                 int count = 0;
```

```
14                      for (int i = 0; i < w.length(); i++) {
15                              if (w.charAt(i) == a) {
16                                      count++;
17                              }
18                      }
19                      if (count % 2 == 0) {
20                              return 1;
21                      } else {
22                              return -1;
23                      }
24              }
25      }
```

End of solution.

Exercise 59. Rice's theorem:
Let S be a set of languages, suppose $\exists$ TM $M_Y : L(M_Y) \in S$ and $\exists$ TM $M_N : L(M_N) \notin S$,

(1) show $L_S = \{M | L(M) \in S\}$ is undecidable,

(2) also show assumptions are necessary.

Solution:

(1) (i) Let us assume that $\emptyset \in S$.
Assume D decides L_S. We build D' to decide ATM

$$\begin{aligned} D' \ := \ & \text{"on input } (M, w) \text{ construct machine} \\ & M' := \text{ "on input } w' \text{ run } M \text{ on } w \text{ if } M \text{ accepts } w, \\ & \qquad \text{return } M_N(w')\text{"} \\ & \text{return } D(M')\text{."} \end{aligned}$$

Suppose M accepts w, then $L(M') = L(M_N) \notin S \Rightarrow$ Reject.
Suppose M rejects or loops on w, then $L(M') = \emptyset \in S \Rightarrow$ Accept.

(ii) what if $\emptyset \notin S$
Think of the complement of $S : \bar{S}$, note L_S is decidable $\Leftrightarrow$ $L_{\bar{S}}$ is decidable. Repeat the argument with $L_{\bar{S}}$. Note $\emptyset \in \bar{S}$.

(2) Let's consider what if we drop one of the two assumptions:

- If we drop "$\exists$ TM $M_Y : L(M_Y) \in S$", then $S = \emptyset$, which means $\{M | L(M) \in S\}$ can be decided by a TM that rejects any input at the first step.

- If we drop "$\exists$ TM M_N : $L(M_N) \notin S$", then for any TM M, $L(M) \in S$. Then $\{M|L(M) \in S\}$ can be decided by a TM that accepts any input at the first step.

Apparently we cannot drop them at the same time either. So, both of the assumptions are necessary for the statement to be true.

End of solution.

Lecture 11

11.27 Summary

In this lecture, we continue discussing the topic of Turing machine, decidable and undecidable problems, and give a sketch proof of Hilbert's tenth problem.

11.28 Exercises

Exercise 60. 2PDA is a PDA with an extra stack (in total 2 stacks). Prove that Accept 2PDA is undecidable.

Solution:

Proof. Sketch: we first show that 2PDA can simulate a TM, and second use "ATM is undecidable ⇒ A2PDA is undecidable"

We first push two special symbols \$ to two stacks to mark the bottoms. After that, PDA reads the entire input and push it symbol by symbol to the second stack. And then all these symbols are moved to the first stack. The motivation behind is we want to use the first stack to represent the portion to the right of the tape head while use the second stack to represent the portion to the left of the tape head. When tape head moves right, we pop one symbol from stack one and push it into stack two. Similarly, we pop one symbol form stack two and push it into stack one to simulate tape head moving left. Note when stack one only has a symbol \$ and tape head will move right, we need to push a blank here since the head is beyond the right margin.

Therefore, we can see that a 2PDA can totally simulate operations of a TM. If A2PDA is decidable, then ATM is decidable. However, we know that ATM is undecidable ⇒ A2PDA is undecidable. □

End of solution.

Exercise 61. QDA is a DFA with a queue. Prove that Accept QDA is undecidable.

Solution:

Proof. Sketch: similar to the last one, but show QDA can simulate TM.

We use two special symbols, say \$ and ϕ to help us to simulate the left and right moves of a TM through a queue. After we start the simulation, we first push ϕ and then the entire input into the queue. Clearly, ϕ stands for the leftmost position of the tape. When the tape head moves right, we pop the top of the queue, overwrite it if necessary, and push this symbol to the end of the queue (including the special symbol ϕ if the tape head moves right from the first symbol). For a left move, push \$, and keep popping and pushing symbols until \$ is found. Then we know that the previous popped symbol is the one to the left of the tape head.

Therefore, we can see that a QDA can totally simulate operations of a TM. If AQDA is decidable, then ATM is decidable. However, we know that ATM is undecidable $\Rightarrow$ AQDA is undecidable. $\square$

End of solution.

Exercise 62. Show $\forall L$
L is turing recognizable $\Leftrightarrow \exists$ decidable language L' where $L = \{w | \exists y : \langle w, y \rangle \in L'\}$

Solution:

Proof. $\Rightarrow$
Suppose L is recognizable and $L(M) = L$ meaning M is a machine that recognizes the language L. We can define $L' = \{\langle w, y \rangle | M$ accepts w in y steps$\}$. Obviously this is decidable because we can run M for y steps and if it accepts we accept, otherwise we reject. This computation must halt with an answer in a finite number of steps, so it is decidable.

Proof. $\Leftarrow$
Suppose there is a decidable language L' thus $\exists$ a machine $M' : L(M') = L'$ meaning M' decides L'. We can use it to construct machine M which recognizes L.

```
Define M :=
  on input w
    For each string y in order
     if M' accepts <w,y> then accept
```

Thus M will accept L within a finite amount time so L is recognizable

End of solution.

Exercise 63. GOTO programs (for register machines)
Machines have a finite number of registers $R_1, R_2 ... R_n$, each of which hold an integer.
Possible instructions are:

- inc R_j
- dec R_j
- goto l
- if $R_j = 0$ goto l
- halt

Simulate $R_i = R_j$ with a GOTO Program

Solution:
1: If $R_j = 0$ goto 5
2: dec R_j
3: inc R_0
4: goto 1
5: if $R_0 = 0$ goto 10
6: inc R_j
7: inc R_i
8: dec R_0
9: goto 5
10: halt

End of solution.

11.29 Hilbert's tenth problem

$H10 : \{p(x_1 \dots x_n) | \exists a_1 \dots a_n$ such that $p(a_1 \dots a_n) = 0\}$
We now give main idea that $H10$ is undecidable.

Register machine is a machine with finite number of registers $R_1, R_2 \dots$ each holding an integer.
GOTO program for register machine has these instructions:
INC R_j : increments register R_j by 1
DEC R_j : decrements register R_j by 1
GOTO l : goto line l
IF $R_j = 0$ GOTO l : goto line l if register R_j is 0
$HALT$: ends the program

Example for a GOTO program looks like this:
1: INC R_0
2: IF $R_0 = 0$ GOTO 5
3: DEC R_0
4: GOTO 2
5: HALT

Define $H_{GOTO} := \{G : G$ is a GOTO program that halts when started with $R_j = 0, \forall i\}$

Fact: H_{GOTO} is undecidable.
We reduce H_{GOTO} to $H10$, i.e. given a GOTO program, we compute a polynomial P:
$P \in H10 \Leftrightarrow G \in H_{GOTO}$
Firstly, we actually work with systems of polynomials. i.e. given $P_1, P_2, \dots P_l$ do they have

a common solution?

Question: How do you reduce a system to a single polynomial?
$P_1, P_2 \Rightarrow P_1^2 + P_2^2$

Variables: We will encode tuples in base $B = 2^b$ (which is large enough)
W_j : contents of R_j through computation
N_l : indicators of the time when instruction l is executed
Example: $W_0 = (0, 0, 0, 1, 1, 0)$
$N_2 = (0, 1, 0, 0, 1, 0)$

We need to express $x \lhd y$. $x \lhd y$ indicates every 'bit' of 'y' is less than or equal to that 'bit' of 'x'.
Example:
$1001 \lhd 1101$
$11 \not\lhd 01$
Fact: $x \lhd y \Leftrightarrow \binom{y}{x}$ is odd, and latter can be written as a pollynomial.
$T = \Sigma B^i = (1, 1, 1, 1, 1, 1, 1, 1, 1)$

The program starts with the first instruction $1 \lhd N_i$
For instruction i: GOTO l
We can have $B * N_i \lhd N_l$

For instruction i: INC R_j
We can have $B * N_i \triangle N_{i+1}$
$W_j = B * (W_j + \Sigma_{i:INCR_j} N_i - \Sigma_{i:DECR_j} N_i)$

For instructioni: IF $R_j = 0$ GOTO l
We can have $B * N_i \lhd N_l + N_{i+1}$
$B * N_i \lhd N_{i+1} + B * T - 2W_j$

Lecture 12

The first exercise is about Turing machines that print or enumerate the strings of a language.

Exercise 64. Say that a TM is a *printer* if it takes no input and prints strings. A language is *printable* if there exists a printer TM that prints it. Prove that for every language L

1. L is Turing-recognizable if and only if it is printable; and
2. L is decidable if and only if it is printable in lexicographic order.

Solution: We begin by proving both directions of the first part. If L is Turing-recognizable then there is some Turing machine M that recognizes it. Let the strings w of L be indexed from 1 to $|L|$. We can construct a TM P that prints L in the following way.

1. No input. Let $i = 1$.
2. For each j in the range $1 \leq j \leq \min\{i, |L|\}$,
3. Run M for i steps on input w_j.
4. If M is in an accept state, print w_j.
5. Increment i and go to stage 2.

Clearly P prints exactly those strings of L that M accepts in any finite number of steps. Since M recognizes L, it follows that P prints L and L printable.

For the other direction, if L is printable then there is some TM P that prints it. We can construct a TM M that recognizes L in the following way. On input w, run P. If P ever prints w, then *accept*. The set of strings that M accepts is L, so L is Turing-recognizable.

To prove the second part, first suppose L is decidable. Then there is some TM M that decides it. Construct a TM P that enumerates all strings in Σ^* in lexicographic order. Say P runs M on each string w and prints w only when M accepts. Then we have that L is printable in lexicographic order.

If L is printable in lexicographic order, then there is a TM P that prints it in lexicographic order. If L is infinite, then for any string $w \in L$ there is a string $w' \in L$ that comes after w in lexicographic order. In this case, we can construct a TM M that decides L in the following way. On input w, run P. If P prints w, *accept*; if P prints w' that comes after w in lexicographic order, *reject*. If L is finite, then we can simply construct a TM M that encodes L as a table and checks whether an input w is in the table.

This distinction is necessary because if L is finite, there is no guarantee that P will ever print a string w' that comes after a given w in lexicographic order, and we have to *reject* if $w \notin L$, not loop forever. In either case, we have a TM M that halts on all inputs, accepting all strings in L and rejecting all strings not in L. This shows L is decidable.

End of solution.

Let $x, y \in \{0,1\}^n$. How can we represent the pair $\langle x, y \rangle$ as a string over $\{0,1\}$ unambiguously, i.e., such that we can recover x and y from the string alone? We can't simply concatenate x and y, because there would be no way to know where x ends and y begins. One solution is to duplicate each bit in x and insert the delimiter 01 before y:

$$\langle x, y \rangle = x_1 x_1 x_2 x_2 \cdots x_{|x|} x_{|x|} 01 y$$

The length of this representation is $|\langle x, y \rangle| = 2|x| + |y| + c$, where c is a constant. A more efficient solution is to begin the string with the length of x as a binary number, encoded as before with each bit duplicated, and then to insert the delimiter 01 before xy. The length of this representation is $|\langle x, y \rangle| = 2 \lg |x| + |x| + |y| + c$.

Exercise 65. Do better.

Solution: We can do slightly better by repeating the strategy and encoding the length of the length of x with duplicated bits, and following that with 01, the length of x, and finally xy. This representation gives $|\langle x, y \rangle| = 2 \lg \lg |x| + \lg |x| + |x| + |y| + c$.

End of solution.

For a string $x \in \{0,1\}^*$, we'll define a *description* of x as an encoding $\langle M, w \rangle \in \{0,1\}^*$ of a TM M and string w such that M halts with x on its tape when run on input w. The *Kolmogorov complexity* $\mathrm{K}(x)$ is the length of the shortest description of x. We say that a string x is *incompressible* if $\mathrm{K}(x) \geq |x|$.

Fact 33. $\exists c. \forall x. \mathrm{K}(x) \leq |x| + c$.

Proof. Construct a TM M that immediately halts on its input. Then $\langle M, x \rangle$ is a description of x that is only greater in length than x by some constant-length encoding of M. □

Fact 34. $\exists c. \forall x. \mathrm{K}(xx) \leq \mathrm{K}(x) + c$.

Proof. Let $\langle M, w \rangle$ be a minimum-length description of x. Construct a TM M' that runs w on M to produce x, and then outputs two copies of x. Then $\langle M', w \rangle$ is a description of xx that is only greater in length than $\langle M, w \rangle$ by some constant-length encoding of M'. □

Fact 35. $\exists c. \forall x, y. \mathrm{K}(xy) \leq 2 \lg \mathrm{K}(x) + \mathrm{K}(x) + \mathrm{K}(y) + c$.

Proof. Let $\langle M, w \rangle$ and $\langle M', w' \rangle$ be minimum-length descriptions of x and y, respectively. Construct a TM N that runs M on w and M' on w' to produce x and y, and then outputs xy. We can encode M by duplicating the bits of $\langle M, w \rangle$ and inserting a delimiter before $\langle M', w' \rangle$. This gives a description of xy not greater than $2 \lg \mathrm{K}(x) + \mathrm{K}(x) + \mathrm{K}(y) + c$. □

Fact 36. There is an incompressible string of length n for all n.

Proof. Recall the number of strings of length n is $|\{0,1\}^n| = 2^n$, and the sum of the number of strings of length $i = 0$ to $n - 1$ is $\sum_{i=0}^{n-1} 2^i = 2^n - 1$. If all strings of length n were incompressible, then there would have to be a string of length less than n describing each string of length n. But $2^n > 2^n - 1$, so some string of length n cannot be described by any possible string of length less than n. Thus there is an incompressible string of length n. □

Exercise 66. Prove $\exists c.\, \forall x.\, \mathrm{K}(x^2 + 17) \leq \mathrm{K}(x) + c$.

Solution: Let $\langle M, w \rangle$ be a minimum-length description of $x \in \{0,1\}^*$. We can construct a TM N that reads a number y in binary as input and halts with the binary representation of $y^2 + 17$ as output. Note that $|\langle N, y \rangle|$ is some constant-length description of N plus y. We can construct a TM M' that runs M on w to produce x, and then runs N on x to produce $x^2 + 17$. Thus $\langle M', w \rangle$ is a description of $x^2 + 17$ that is only greater than $\langle M, w \rangle$ in length by some constant; the minimum-length description $K(x^2 + 17)$ must be no more than $K(x) + c$ for some c and any x.

End of solution.

Exercise 67. Prove that the set of incompressible strings is undecidable.

Solution: Assume the set of incompressible strings is decidable. Recall from Exercise 64 that a language is decidable iff there is a TM P that prints each string in the language in lexicographic order. Therefore we can construct a TM P' that, on input n, runs P until it prints an incompressible string of length n and then halts with that string as output. We also know from Fact 36 that there is an incompressible string of length n for every length n, which means P' halts on all n. Therefore $\langle P', n \rangle$ is a description of some incompressible string x of length n, i.e., $\mathrm{K}(x) \geq n$, and moreover $|\langle P', n \rangle| \geq n$.

But note that $\langle P', n \rangle$ is just some constant-length encoding of P' plus $\lg n$ bits encoding n in binary. That is, $|\langle P', n \rangle| = \lg n + c$. Clearly $n > \lg n + c$ for any constant c and sufficiently large n, so we have a contradiction. This shows our initial assumption was false.

End of solution.

Exercise 68. Prove that $\mathrm{K}(x)$ is not computable.

Solution: Assume $\mathrm{K}(x)$ is computable. Then there is a TM M that, on input x, halts with $\mathrm{K}(x)$ as its output. Further, we can construct a TM M' that, on input x, runs M on x to produce $\mathrm{K}(x)$, and then *accepts* if $\mathrm{K}(x) \geq |x|$ and *rejects* otherwise. But M' is a decider for the set of imcompressible strings, which contradicts Exercise 67. Therefore our initial assumption was false, and $\mathrm{K}(x)$ is not computable.

End of solution.

Exercise 69. Prove $\forall c.\, \exists x, y.\, \mathrm{K}(xy) > \mathrm{K}(x) + \mathrm{K}(y) + c$.

Solution: In class; to appear below.

End of solution.

Lecture 13

13.30 Summary

We covered material from slides-complexity.pdf, first reviewing the material covered before Spring Break and then proceeding to the definition of P and several examples of reductions of 3SAT onto various other problems.

13.31 New Scribes Rules

For the rest of the semester, the scribes-exercises.tex updates will be handled as follows. We are using Subversion to manage versions of each file, and we should avoid sending multiple e-mails to the class for successive versions of the same lecture notes. Instead, we should e-mail Emanuele when we are ready for him to review our work, and only e-mail the class with the final version. If we wish, we can simply commit the final version to Subversion and notify the class when it's done.

The group responsible for a given lecture should continue working on the problems until they are solved, even if this continues past the due date for the notes.

Finally, Emanuele will go over the completed scribes notes at the beginning of class and comment on the solutions provided.

13.32 Complexity: P

We reviewed the material from last time, and continued through the reduction of 3SAT onto SUBSET-SUM. All the notes today were in the slides, but here are some highlights:

- A language which is only decidable in some huge amount of time is useless. We analyze this by calculating the runtime for a TM as a function of the size of the input. For some input w, we typically denote $|w|$ as n.
- The runtime of an algorithm depends to some extent on the language in which it's implemented. In particular, the locality of Turing Machines prevents them from being able to access arbitrary memory locations in constant time. Therefore, many algorithms which are linear (say) in Java are quadratic on a TM. However, an algorithm which is polynomial in Java is still polynomial on a TM.
- We define the language $\text{TIME}(t(n)) = \{L : L \text{ can be decided by a TM that runs for at most } t(n) \text{ steps on an input of size } n\}$.
- We define $P = \bigcup_c \text{TIME}(n^c)$.
- This class studies what is NOT in P. It turns out that about the only thing we can prove is not in P is ATM (and other undecidable languages), and about the only kind of proof which has been shown to demonstrate that a language is not in P is the proof for ATM's undecidability.

- There is a class of interesting languages known as NP-complete, which are not believed to be in P. These languages are interesting partly because if any one of them is ever shown to be in P, then all of them must be in P. They are also interesting because they have a lot of applications for business and science.
- We explain the concept of reduction: for two languages A and B, if we can prove that $A \in P \implies B \in P$ then we call that proof a reduction of B to A.
- We went over several algorithms in NP-complete, including 3SAT, CLIQUE, and SUBSET-SUM and saw reductions of 3SAT to the latter two.

13.33 Exercises

Exercise 70. Find a Turing Machine R which on input ϕ computes graph G_ϕ and integer t_ϕ such that $\phi \in$ 3SAT $\iff (G_\phi, t_\phi) \in$ CLIQUE

Solution: Since there is no limitation on the running time of R, it can simply try all possible assignments for ϕ to find a satisfying assignment. If there is one then R can choose some arbitrary t_ϕ and construct a complete graph G_ϕ with t_ϕ nodes. If there is no satisfying assignment for ϕ then R can return the same G_ϕ with a t_ϕ equal to one more than the number of nodes in the graph. (G_ϕ, t_ϕ) is in CLIQUE if and only if ϕ has a satisfying assignment.
End of solution.

Exercise 71. Find a polynomial time reduction of 3SAT to INDEPENDENT-SET.

Solution: This proof is very similar to the reduction in the slides of 3SAT to CLIQUE. The only difference is the manner in which we select our edges.
INDEPENDENT-SET is defined as $\{(G, t) : G$ is a graph containing an independent set of size $t\}$.

For any 3CNF ϕ, we define G_ϕ and t_ϕ as follows. t_ϕ is simply the number of clauses in ϕ. $G_\phi = (V, E)$ is a graph with one vertex per variable in ϕ, so $|V| = 3t_\phi$. We add edge (u, v) to E if vertices u and v are in the same clause or if they are logically inconsistent (e.g. x and $\neg x$).

Claim 37. $\phi \in$ 3SAT $\iff (G_\phi, t_\phi) \in$ INDEPENDENT-SET

Proof. $\Rightarrow$ Suppose there is some variable assignment which satisfies ϕ. That means that there must be at least one variable in each clause which can be simultaneously satisfied, so these variables are not logically inconsistent. Since the variables are in separate clauses and logically consistent, we must not have edges connecting their corresponding vertices in G_ϕ. Therefore, we have an independent set of size t_ϕ in G_ϕ.

$\Leftarrow$ Suppose there is an independent set of size t_ϕ in G_ϕ. The vertices in this set must not have come from the same clause, and they must not be logically inconsistent: otherwise, they would have an edge between them. Therefore, these vertices correspond to a valid assignment of variables which satisfies ϕ. $\square$

We can evaluate the runtime of this reduction by examining the size of the data structure $G_\phi = (V, E)$. There is one vertex per variable, so $|V| = O(|\phi|)$. There can be at most a quadratic number of edges, so $|E| = O(|\phi|^2)$. Since $|\phi|^2+|\phi| \in P$, this reduction is polynomial time.

End of solution.

Exercise 72. Find a polynomial time reduction of 3SAT to SYSTEM.

Solution: This reduction can be easily achieved using the following approach.

For each variable x in the 3SAT problem ϕ, we will add two variables x_t and x_f to the system problem S representing the possible assignments to x. We will also add the following constraints to S: $0 \leq x_t \leq 1$, $0 \leq x_f \leq 1$, and $x_t + x_f = 1$. Note that these constraints force x_t and x_f to act like Boolean variables with opposite values.

For each clause $(x_1 \vee x_2 \vee x_3)$ in the 3SAT problem we will also add the constraint $x_1 + x_2 + x_3 \geq 1$, using the appropriate variables x_t and x_f for the clause.

Claim 38. $\phi \in$ 3SAT $\iff S \in$ SYSTEM

Proof. $\Rightarrow$ Suppose there is some variable assignment which satisfies 3SAT. That means that there must be at least one variable x_t or x_f in each clause which equals 1, so the sum for the clause constraint $x_i + x_j + x_k \geq 1$ will be satisfied. Thus, we have a satisfiable assignment for the corresponding SYSTEM.

$\Leftarrow$ Suppose there is a variable assignment for $S \in$ SYSTEM such that $0 \leq x_i, x_j, x_k \leq 1$ and $x_i + x_k + x_j \geq 1$. Since we have integer assignments, we can conclude that at least one of these variables is equal to one. We can set the corresponding variable in ϕ to either true or false depending on whether the variable which is 1 is a x_t variable or a x_f variable and satisfy the clause in ϕ. Since every constraint in S is simultaneously satisfied, every clause in ϕ is also satisfied. Therefore, these integer assignment can lead to a valid answer to the equivalent 3SAT problem. $\square$

Our reduction converts each variable to a constant number of constraints in $O(|\phi|)$ time, and converts each clause to the corresponding linear inequality in $O(|\phi|)$. This gives us a polynomial runtime of $O(|\phi|)$ for the reduction.

End of solution.

Lecture 14

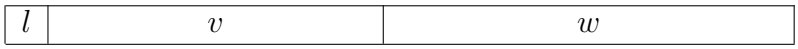
14.34 Summary

In this lecture, we finished one more example of reduction of 3SAT onto 3COLOR, proved all the covered examples are in NP, closed the NP circle and introduced EXP.

14.35 Exercises

Exercise 73. Prove the claim $\forall c\ \exists x, y : K(xy) \geq k(x) + k(y) + c$

Proof. High-level idea: We note that a string describes its own length. Let Z be an incompressible string of length at least n. We will try to break Z into xy so that it satisfies the claim.

$Z =$ 

Suppose $|Z| = n$, $|l| = \frac{log(n)}{2}$, $|v| = 1l$ (in binary) and $|lv| = 1l + \frac{log(n)}{2}$. That is, we call Z's first $\frac{log(n)}{2}$ bits l, and interpret $1l$ as the length of v.

Then consider the number in binary and we get $1l \in \{2^{\frac{log(n)}{2}}, \ldots, 2^{\frac{log(n)}{2}+1} - 1\}$. In other words, $1l = \theta(\sqrt{n})$. The Kolmogorov complexity of v satisfies $K(v) \leq |v| + c$. Next, we construct a TM on input v that computes as follows:

1. First compute on(record) v
2. Then compute $|v| = 1l$
3. Output lv

Hence, $K(lv) \leq |v| + c$. Also, we have $K(w) \leq |w| + c$ and $K(lvw) \geq n$. Therefore $K(lv) + K(w) \leq |v| + |w| + c = n - \Omega log(n)$. $\square$

Exercise 74. Without using the Cook-Levin theorem, prove that the following language is NP-complete: $\{(M, x, 1^t) : M$ is a Turing machine: $\exists y \in \{0,1\}^t : M(x, y)$ accepts within t time steps$\}$.

Solution: We call this language L. We first show that L $\in$ NP, then we prove that if L $\in$ P, P = NP.

1. Input $w = (M, x, 1^t)$, $y \in \{0,1\}^t$, $|y| \leq |w|$, construct a Turing machine M', which checks if $M(x, y)$ accepts within t steps. M' will check no more than $t \leq w$ steps. This will take polynomial time for M', so $L \in NP$.
2. For any NP language A, there is an integer c and a Turing machine M_A that runs in polynomial time n^c such that $w \in A \iff \exists y, |y| \leq |w|^c, M_A$ accepts (w, y).

We construct $w' = (M_A, w, 1^{n^c})$, and have:
$w \in A \iff \exists y, |y| \leq |w|^c, M_A$ accepts (w, y) within n^c steps $\iff w' \in$ L.
As the reduction from w to w' is polynomial, we can conclude that any language in NP is polynomial time reducible to L. If L is in P, then P = NP.

End of solution.

Exercise 75. Suppose P=NP, give an algorithm that on input a satisfiable 3CNF φ, outputs a satisfying assignment in time poly($|\varphi|$).

Solution: Let the set of variables in φ be $x_1, x_2, \cdots, x_k$

Algorithm 1 Find-Assignment(φ)

```
Array A[1 ··· k]
for i = 1 to k do
  φ0 ← φ ∧ (x̄_i ∨ x̄_i ∨ x̄_i)
  φ1 ← φ ∧ (x_i ∨ x_i ∨ x_i)
  if φ0 is satisfiable then
    φ ← φ0 A[i] ← 0
  else if φ1 is satisfiable then
    φ ← φ1 A[i] ← 1
  else
    REJECT
  end if
end for
return A
```

If P=NP, then there exists a polynomial time algorithm T that determines whether a 3CNF is satisfiable or not. For each variable x_i we construct φ_0 and φ_1 which constrain x_i to a particular value. We call T to find out whether φ_j is satisfiable. If it is, we remember our choice in array A and replace φ with the satisfiable φ_j so future assignments will have to work with past decisions. Since we call T k times, the running time of our algorithm is still polynomial.

End of solution.

Exercise 76. Suppose that there exists a decider for 3SAT that runs in time $t(n)$. Describe an algorithm that you can actually run that provides a satisfying assignment for 3SAT with running time $t^c(n)$, for some constant c.

Solution: From exercise 75 we know that if there exists a decider for 3SAT that runs in time $t(n)$, there is an algorithm that outputs a satisfying assignment whose running time is $t^c(n)$. We go through all algorithms in some sorted order in a brute force search to find a correct decider. At step 1, we run algorithm 1 for one step; at step 2, we run algorithm 1

one step further, and run algorithm 2 for 2 steps; at step 3, we run algorithm 1 one step further, run algorithm 2 one step further, and run algorithm 3 for 3 steps, and so on.

Since we know that the input is satisfiable, and by the assumption that some correct algorithm exists, at some step i we will get a satisfying assignment. Note that the value of i does not depend on the size of the input: it can be treated as a constant. At that point, we will have run i algorithms for i steps each. The algorithms which output possible satisfying assignments will be verified in polynomial time, with at most i total verifications run. Therefore, the total running time will be $O(i^2 + it^c(n))$ for some c.

End of solution.

Lecture 15

15.36 Summary

In this lecture, we started with proving $P \neq EXP$. We finished the proof of Cook, Levin Theorem and then we covered Interactive Proof Systems section till Graph Labeling.

15.37 Exercises

Exercise 77. Prove Quadratic Programming $\in$ P $\Rightarrow$ 3SAT $\in$ P.

Solution:

On $\forall$ input $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_i \vee b_i \vee c_i) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$ of 3SAT, generate R(ϕ), set of quadratic conditions as follows:

For each literal x, $x + \neg x = 1 \quad x^2 + (\neg x)^2 = 1 \quad x, \neg x \geqq 0$

For each clause $(a_i \vee b_i \vee c_i)$, $a_i + b_i + c_i \geqq 1$

Claim 39. $\phi \in$ 3SAT $\iff$ R(ϕ) $\in$ Quadratic Programming

Proof. $\Rightarrow$ Satisfying assignment of 3SAT will satisfy all the quadratic conditions. Trivial.

$\Leftarrow$ All literals should have value of 0 or 1 from first set of quadratic conditions and all negating literals, e.g. $x, \neg x$, should have different value. Therefore, if there is a set of literals that satisfy Quadratic programming conditions, we can convert any literal with value of 1 to True and 0 to False, and that should satisfy 3SAT.

The conversion generates constant factor more equations compared to the length of input. □

End of solution.

Exercise 78. Reduce 3SAT to Hilbert's tenth problem in polynomial time.

Solution: We give TM R that on input ϕ

- computes a polynomial P such that $\phi \in$ 3SAT $\Leftrightarrow$ P has an integer solution.
- runs in polynomial time.

Definition of R:
On input $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$,

- compute

$$\begin{aligned}\neg\phi =& (\neg a_1 \wedge \neg b_1 \wedge \neg c_1) \vee (\neg a_2 \wedge \neg b_2 \wedge \neg c_2) \vee \ldots \vee (\neg a_k \wedge \neg b_k \wedge \neg c_k) \\ =& (d_1 \wedge e_1 \wedge f_1) \vee (d_2 \wedge e_2 \wedge f_2) \vee \ldots \vee (d_k \wedge e_k \wedge f_k).\end{aligned}$$

Here d_i, e_i, f_i are literals and $\neg\phi$ is in disjunctive normal form.
For example, if $\phi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$, then

$$\begin{aligned}\neg\phi =&(\neg x \wedge \neg y \wedge \neg z) \vee (\neg\neg x \wedge \neg\neg y \wedge \neg z)\\ =&(\neg x \wedge \neg y \wedge \neg z) \vee (x \wedge y \wedge \neg z).\end{aligned}$$

- Construct a polynomial P containing all the variables in $\neg\phi$.

  For each clause in $\neg\phi$, construct a quadratic term in P by

  - replacing "$\wedge$" with "$\times$",
  - replacing "$\neg$" with "1-",
  - squaring the term.

  And P=sum of all these quadratic terms.

  For example, if $\neg\phi = (\neg x \wedge \neg y \wedge \neg z) \vee (x \wedge y \wedge \neg z)$, then $P = [(1-x)(1-y)(1-z)]^2 + [xy(1-z)]^2$.

It is easy to see that R runs in polynomial time.
$\Rightarrow$:
If $\phi \in$ 3SAT, there is an assignment such that $\phi = 1$, then $\neg\phi = 0$. We claim that this assignment is an integer solution to P. Because $\neg\phi = 0$, every clause in $\neg\phi$ is 0. So in every clause, there is a literal $= 0$. So every quadratic term in P is 0, and therefore $P = 0$. For example, $(x = 1, y = 0, z = 0)$ is an assignment such that $\neg\phi = (\neg x \wedge \neg y \wedge \neg z) \vee (x \wedge y \wedge \neg z) = 0$. It is also a solution to $P = [(1-x)(1-y)(1-z)]^2 + [xy(1-z)]^2$.
$\Leftarrow$:
If P has an integer solution, we can transform it to an assignment for $\neg\phi$ by changing all variable values which are neither 0 nor 1 to 1. For example, $(x = 1, y = 0, z = -5)$ is a solution to $P = [(1-x)(1-y)(1-z)]^2 + [xy(1-z)]^2$, then we transform it to $(x = 1, y = 0, z = 1)$ as an assignment for $\neg\phi$. We claim that $\neg\phi = 0$ with this assignment. $P = 0$ implies that every quadratic term in P is 0. So for every quadratic term either

- there is a variable $v = 0$, and v appears in that quadratic term and the corresponding clause, or
- there is a variable $v = 1$, and $1 - v$ appears in that quadratic term, $\neg v$ appears in the corresponding clause.

In either case, the corresponding clause $= 0$. So $\neg\phi = 0$. So $\phi = 1$.

End of solution.

Exercise 79. Prove Independent set $\in$ P $\Rightarrow$ Clique $\in$ P.

Solution: For $\forall$ input ϕ of Clique, create a new graph $\neg\phi$ that negates the edges: if $(x, y) \in \phi$, then $(x, y) \notin \neg\phi$, and vice versa. Then $\phi \in$ Clique $\Leftrightarrow \neg\phi \in$ Independent set. Conversion takes linear time compared to the size of input.

End of solution.

Exercise 80. Prove NAE-3SAT $\in$ P $\Rightarrow$ 3SAT $\in$ P.

Solution: On $\forall$ input $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_i \vee b_i \vee c_i) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$ of 3SAT, generate Q(ϕ), input of NAE-3SAT as follows:

For each clause $(a_i \vee b_i \vee c_i)$, generate two clauses $(a_i, b_i, x_i) \wedge (c_i, \neg x_i, \text{“False”})$.

The conversion takes linear time compared to the size of input.

Claim 40. $\phi \in$ 3SAT $\iff$ Q(ϕ) $\in$ NAE-3SAT

Proof. $\Rightarrow$ For any satisfying set of literals, let $x_i = \neg(a_i \vee b_i)$ for each clause.

$\Leftarrow$ Proof by contradiction. If all literals of a clause is False, neither setting x_i to True nor False satisfies both clauses in NAE-3SAT. Trivial. $\square$

Since Q(ϕ) is not a valid NAE-3SAT clause with constant “False”, we substitute it to a global variable α, and Q(ϕ) = $(a_i, b_i, x_i) \wedge (c_i, \neg x_i, \alpha)$

Claim 41. $\phi \in$ 3SAT $\iff$ R(ϕ) $\in$ NAE-3SAT

Proof. $\Rightarrow$ For any satisfying set of literals, set α to be “False” and follow the proof of previous claim.

$\Leftarrow$ Let t be a satisfying NAE-3SAT assignment of R(ϕ). If $\alpha =$ “False” in the assignment, follow the proof of previous claim. If $\alpha =$ “True”, you can find an alternative solution of NAE-3SAT, $\neg t$, which negates all the literals in t and α become “False”. Then, follow the proof of previous claim. $\square$

End of solution.

Exercise 81. Prove MAX-CUT $\in$ P $\Rightarrow$ NAE-3SAT $\in$ P.

Solution: For input $\phi = (a_1, b_1, c_1) \wedge (a_2, b_2, c_2) \wedge \ldots \wedge (a_i, b_i, c_i) \wedge \ldots \wedge (a_l, b_l, c_l)$ of NAE-3SAT, create graph R(ϕ) of MAX-CUT as follows:

For each clause (a_i, b_i, c_i), create a graph where each literal is a node and there is an edge between two literals if they are not identical (i.e. “clause subgraph”). Between “clause subgraph”s, connect nodes that negates each other. For instance, a node x in clause i (or “clause subgraph” i, from viewpoint of R(ϕ)) and node $\neg x$ in clause j gets an edge between. Count the number of inter-clause edges as you generate the graph and let k be the total number of inter-clause edges. The conversion takes quadratic time to create edges between literals.

Claim 42. $\phi \in$ NAE-3SAT $\iff$ (R(ϕ), $k + 2l$) $\in$ MAX-CUT

Proof. $\Rightarrow$ If $\phi \in$ NAE-3SAT, all clause subgraph of R(ϕ) must have at least 2 edges since not all three literals of a clause should be identical. Now, let's bisect the nodes R(ϕ) following the satisfying assignment of ϕ: all nodes with True assignment into one set and all nodes with False assignment into another set, cutting all edges that connects True literals and False literals. Then, two edges of each clause subgraph must be cut since ϕ satisfies NAE-3SAT, and all k inter-clause edges will be cut since edges were created only between nodes that negate each other. Therefore, you can find a cut of $k + 2l$ edges.

$\Leftarrow$ Proof by contradiction. Assume there is no satisfying assignment of NAE-3SAT.

Case 1. There is a clause where all three literals are identical in ϕ. Then, there exists no edge in the "clause subgraph" of three identical literals. Since there are k inter-clause edges and you can cut at most 2 edges per "clause subgraph", you can never find a cut of $k + 2l$ or more.

Case 2. There is no clause where all three literals are identical in ϕ. Then, for any cut that cuts all k inter-clause edges, there must be a "clause subgraph" where all literals have same assignment, True of False, since ϕ does not satisfy NAE-3SAT. Since you cannot cut more than 2 edges per "clause subgraph" and there are at most k inter-clause edges, there is no cut of $k + 2l$ or greater. $\square$

End of solution.

Lecture 16

16.38 Summary

In this lecture, we firstly solved Exercise 76 and then finished the section of Interactive Proof Systems. After that, we continued to learn Zero-knowledge system and cover the contents in `complexity-misc` slides.

16.39 Exercises

Exercise 82. (1) Formalize languages with deterministic, interactive proof systems.

(2) Show such languages are exactly NP.

Solution:

(1) We name this kind of languages as **DIPS** and formalize this language as follows:
Let $V, P : \{0,1\}^* \to \{0,1\}^*$ be functions. A k-round interaction of V and P on input $x \in \{0,1\}^*$ is the sequence of the following strings $a_1, a_2, \ldots, a_k \in \{0,1\}^*$ defined as follows:

$$\begin{aligned} a_1 &= V(x) \\ a_2 &= P(x, a_1) \\ \vdots &\quad \vdots \\ a_{2i+1} &= V(x, a_1, \ldots, a_{2i}) \\ a_{2i+2} &= P(x, a_1, \ldots, a_{2i+1}) \end{aligned}$$

Now we denote $\langle V, P\rangle(x)$ as the output of V at the end of the interaction with P on input x, and obviously a_i is the exchanged information between V and P.

Definition 43. We say that a language L has a k-round deterministic interactive proof system if there is a deterministic Turing Machine V that on input $x, a_1, a_2, \ldots, a_i$ runs in time polynomial in $|x|$, and can have a k-round interaction with any TM P such that:

- $x \in L \Rightarrow \exists P : \langle V, P\rangle(x) = 1$
- $x \notin L \Rightarrow \forall P : \langle V, P\rangle(x) = 0$

The class **DIPS** contains all languages that have a k-round deterministic interactive proof system, where P is polynomial in the input length.

(2) we show that DIPS = NP in the following:

(i) NP $\subseteq$ DIPS: this is true in trivial since each NP language has a 1-round proof system.

(ii) Now we prove that if a L has an interactive proof system of this type then $L \in NP$. The certificate for membership is just the transcript $(a_1, a_2, \ldots, a_k)$ causing the verifier to accept. To verify this transcript, check that indeed $V(x) = a_1$, $V(x, a_1, a_2) = a_3$, ..., and $V(x, a_1, \ldots, a_k) = 1$. If $x \in L$ then there indeed exists such a transcript. If there exists such a transcript $(a_1, a_2, \ldots, a_k)$ then we can define a prover function P to satisfy $P(x, a_1) = a_2$, $P(x, a_1, a_2, a_3) = a_4$, etc. We see that $\langle V, P\rangle(x) = 1$ and hence $x \in L$. Therefore, $L \in NP$.

End of solution.

Exercise 83. Argue in no more than 10 lines that:

$$\begin{aligned} \text{polynomial-time on TM} &= \text{polynomial-time on } k\text{-tape TM} \\ &= \text{polynomial-time on RATM} \end{aligned}$$

Solution:
Simulate a k-tape TM N via a TM M: M uses cells $i, k+i, 2k+i, ...$ to encode the i-th tape of N. For every symbol a in N's alphabet, M also includes $\hat{a}$ to show the position of head. Each step of N is done in polynomial steps in M, so polynomial-time on N will still be polynomial on M.

Simulate a k-tape RATM R with a k-tape TM N: Encode each cell of tapes in pairs like (address, value), where value is the actual value of the original cell. Each random access to a cell can be simulated in this way: scan the whole tape to find the matching address, if find it then read or write the corresponding value, else if not find the address then write a new pair of (address, value). So each random access of R will be done in polynomial steps in N, therefore polynomial-time on R will still be polynomial-time on N.

Simulate a 1-tape TM M via a 1-tape RATM R: Ignore the indexing tape. Done.

End of solution.

Non-deterministic TM: δ maps to subset of $Q \times \Gamma \times \{L, R\}$. It accept if there is a computation path that leads to accept state.

Definition 44. NTIME(t(n)) = {L: L is decided by a non-deterministic TM that runs in time $\leq$ t(n)}.

Definition 45. NTIME(t(n)) = {L: $\exists M$: $\forall x$ of length n s.t. $x \in L \Leftrightarrow \exists y, y \leq$ t(n), $M(x, y)$ accepts in $\leq$ t(n)}.

Exercise 84. Prove the two definitions are equivalent (feel free to use multiple tapes, if that helps).

Solution:
"$\Leftarrow$": Assume $L \in$ NTIME($t(n)$) according to Def2, then $\exists M$ as described in Def2. Construct a non-derterministic TM N as follows.

N = "on input x,
non-deterministically generate a string y with length $\leq t(n)$,
run $M(x, y)$
If it accepts, ACCEPT.
Otherwise, REJECT."

If a string $w \in L$, then N will accept it in time $2t(n)$, because $\exists y$ that can be generated in time $t(n)$ and $M(w, y)$ accepts in time $t(n)$. Else if $w \notin L$, then these is no such a y that makes $M(w, y)$ accept, so N will reject. So $L \in$ NTIME(2t(n)) according to Def1.

"$\Rightarrow$": Assume $L \in$ NTIME($t(n)$) according to Def1, then $\exists$ a non-deterministic TM N as described in Def1. If a string $w \in L$, then N accepts w in time $t(n)$ according to Def1, so there is a computation path that leads to accept state. Encode this computation path using a binary number y: since $|Q \times \Gamma \times \{\text{L,R}\}|$ is a constant, each step of the computation path can be encoded in a constant c bits. So $|y| \leq ct(n)$. We use y_i to denote the ith c bits of y, which encodes the ith step of N's computation. Construct a TM M as follows.

M = " on input (x, y)
run N on x step by step
at step i, y_i indicates which operation to take
If N accepts, ACCEPT.
otherwise, REJECT."

Now we prove that $L \in$ NTIME($t^2(n)$):

If a string $w \in L$, then M will eventually accept (w, y). In M, simulating a step of N takes $O(t(n))$ steps. So, M will accept in $O(t^2(n))$. In addition, $|y| \leq ct(n) \leq t^2(n)$ if n is sufficiently large.

Else if $w \notin L$, then N has no computation path to accept in time $t(n)$. So there is no such a y that M accepts (w, y) in time $O(t^2(n))$.

Therefore, $L \in$ NTIME($t^2(n)$) according to Def2.

End of solution.

Exercise 85. Prove the two definitions are equivalent, up to a small change in t(n) (no more than $t \longrightarrow t^2$).

Solution: As shown in Exercise 83, the two definitions are equivalent up to a small change in t(n). If one is t(n), the other is at most $O(t^2(n))$.

End of solution.

Lecture 17

17.40 Summary

Exam 2.

17.41 Exercises

Exercise 86. Let M be a Turing machine. Show that the language
$\{C\#D^R :$ configuration C does not yield configuration $D\}$
is context-free. Recall that D^R is the reverse of D.

FAQ: What if C or D^R are ill-formed strings that do not correspond to configurations, such as 0011 or $0q_711q_5q_5q_3$? Answer: you are free to treat those any way you like.

Solution: Sketch: We take advantage of the power of CFG as well as locality of TM computation, and show that for TM M if its configuration C_i (as C in this problem) does not yield C_{i+1} (as D in this problem), we can have $C_i\#C_{i+1}^R$ is CFG.

We write the TM computation as: $C_1\#C_2^R\#C_3\#C_4^R\ldots$, and build a CFG G_C such that: $L(G_C) = \Delta^*abc(\Delta - \{\#\})^t\#(\Delta - \{\#\})^tfed\Delta^*$ for any $t \geq 0$ and any 6 symbols

$$\begin{array}{ccc} a & b & c \\ d & e & f \end{array}$$

that are inconsistent with TM transition function δ. This CFG indeed exists and can be built based on the CFG for $w\#w^R$ shown in the class. We detail it here: Let $G_C = (V, \Sigma, R, S)$ be a CFG:

V is a finite set of variables, $V = \{S, A, B, C\}$, and S is the start variable
Σ is a finite set of terminals and $\Sigma = \{x, y, a, b, c, d, e, f\}$,
where $y, a, b, c, d, e, f \in \{\Delta\}, x \in \{\Delta - \#\}$,
Δ is the alphabet for the configuration.

R is based on the following rules:

$S \to yS|Sy|C$ Remark: $S \Rightarrow^* \Delta^*abc(\Delta - \{\#\})^t\#(\Delta - \{\#\})^tfed\Delta^*$
$C \to abcBfed$ Remark: $C \Rightarrow^* abc(\Delta - \{\#\})^t\#(\Delta - \{\#\})^tfed$
$B \to ABA|\#$ Remark: $B \Rightarrow^* (\Delta - \{\#\})^t\#(\Delta - \{\#\})^t$
$A \to x|\varepsilon$

End of solution.

Exercise 87. Let $\Sigma = \{a, b\}$ and consider the language $\Sigma^k a \Sigma^*$, for a constant k. Show that it cannot be decided by a DFA with $\leq k$ states. Hint: one way to solve this is to think of the proof of the pumping lemma.

Solution: We prove by contrapositive. Suppose it can be decided by a DFA with $\leq k$ states. For a string $w = b^k a$ in language $\Sigma^k a$, we know it contains $k+1$ symbols. Inspired by the proof of pumping lemma, and using pigeonhole theory, we are aware of that with $\leq k$ states and $k+1$ symbols, there must be some repeat states $q*$ during computing $b^k a$ by the DFA. Suppose the initial state to compute $b^k a$ is q_0 and the state to accept it is q_a (assume $q_a \neq q^*$), we then have:

$$q_0 \to \ldots \to \underbrace{q^* \to \ldots \to q^*}_{\text{self-loop}} \to \ldots \to q_a.$$

Let y be the label on the self-loop from q^* to q^*. If the self-loop part can be repeated once, then it can be repeated many times. Therefore, we can pump more b from w, for example, $b^{k+|y|}a(|y| > 0)$, and it also can be computed by the DFA. Therefore, any string w' in language $b^{k+|y|}a\Sigma^*$ can be accepted by this DFA. However, w' is not in language $\Sigma^k a \Sigma^*$. This conflicts with the assumption that DFA decides $\Sigma^k a \Sigma^*$.

End of solution.

Exercise 88. Let x be uniform in $\{0,1\}^n$, and let K be Kolmogorov complexity.
Show $\Pr_x[K(x) \geq n-1] \geq 1/2$.

Solution: There are $2^0 + 2^1 + 2^2 + ... + 2^{n-2} < 2^{n-1}$ unique strings of size $< n-1$. We need to represent 2^n strings altogether. Thus $Pr_x[K(x) < n-1] < \frac{2^{n-1}}{2^n}$ or, equivalently, $Pr_x[K(x) \geq n-1] \geq 1/2$

End of solution.

Exercise 89. Let L and L' be two Turing-recognizable languages. Suppose that $L \cap L'$ and $L \cup L'$ are decidable. Show that L and L' are decidable.

Solution: Since L and L' are recognizable $\exists$ TM M and M' that recognize them. Since $L \cap L'$ and $L \cup L'$ are decidable $\exists$ deciders $D_\cap$ and $D_\cup$ that decided them. We will show how to build a decider D_L which decides L (the decider for L' is almost identical.

$D_L :=$ on input w
1) Run $D_\cup(w)$
reject if reject
2) Run $D_\cap(w)$
accept if accept
3) Run (M, w) and (M', w) simultaneously, doing one step of each machine
accept if M accepts

reject if M' accepts

D_L works in the following manner. 1) If w is not in $L \cup L'$, then it can't be in L or L' and we reject. 2) If w is in $L \cap L'$, then it must be in both L and L' and we accept. 3) If we have not accepted or rejected at this point then we know that w is in L or L', but not both. Since L and L' are recognizable either M or M' must accept within a finite number of steps. We run each (M, w) and (M', w) and accept or reject accordingly. Thus, D_L decides L and L (as well as L') is decidable.

End of solution.

Exercise 90. A vertex-cover of size k of a graph is a set C of k nodes such that every edge in the graph touches at least one node in C. Show that $\{(G, k)|$ graph G has a vertex cover of size k $\}$ is NP-complete. Hint: reduce from 3-SAT. For every variable x construct a 2-node graph with nodes x and $\neg x$; for every clause construct a triangle; connect equal nodes.

Solution: Reduce from 3SAT. Construct the graph as mentioned in hint.
If there is a satisfying assignment in 3SAT, we pick the true literals into our vertex cover. By choosing one node for each literal, we can cover the edges between literals in 2-node graphs. Note that 3SAT is satisfied, we can also cover at least an edge between 2-node graph and each triangle. Now we need 2 nodes (or less) in each triangle to cover the rest of edges. We can pick any 2 nodes in each triangle as long as the left node in triangle is the true literal in 3SAT assignment. Assume there are n variables, m clauses, vertex-cover size $k = n + 2m$.
Reverse direction:
Assign value true to literials whose vertex in 2-node graph is in vertex cover. Since $k = n + 2m$, for each clause at least one edge connecting its triangle to 2-node graphs is covered by a literial vertex. We have at least one node in each triangle whose value is true so the value of each clause is true. Thus, we have found a satisfying assignment to 3SAT.

End of solution.

Exercise 91. Let H10 be Hilbert's 10th problem of determining if a given polynomial in multiple variables has an integer solution.
(1) Is H10 $\in$ NP?
(2) Find the flaw in the following proof that H10 $\in$ NP: the witness is the integer solution, it takes polynomial-time to check it.

Solution:
(1)We know from the exercises and slides that H10 $\in$ undecidable, and NP $\subset$ decidable. Therefore, we can infer that H10 $\notin$ NP.
(2) The definition for NP is:
$\{L : \exists$ integer $c, \exists$ TM that runs in time n^c: $w \in L \Leftrightarrow \exists y, |y| \leq |w|^c, M$ accepts $(w, y)\}$
Since the witness integer $y = (y_1, y_2, \ldots, y_n)$ can be arbitrarily large, the length of witness $|y|$ may not be polynomial, which may violate the definition in NP $|y| \leq |w|^c$ so we can not

use the TM M that is defined in NP to accept the input w in poly time.

End of solution.

Exercise 92. Show that the following language is undecidable:
$\{(M, M') : M$ and M' are TM and $L(M) = L(M')\}$.

Solution:

Suppose there exists a decider for language $L(M, M')$, named D.
Construct a decider $D'(M, w)$
Construct TM M_1
    on input x
    run M on w
    if M accepts w, return ACCEPT
    else return REJECT
Construct TM M_2
    on input x
    return ACCEPT
return $D(M_1, M_2)$

Turing Machine M_1 accepts an input if Turing Machine M accepts input w. Turing Machine M_2 accepts all the languages. From the definition of D', we know that "M accepts $w \Leftrightarrow D'$ accepts (M, w)". However, the fact is that ATM is not decidable. Therefore, we can infer that D' does not exist and the assumption that language $L(M, M')$ is decidable is incorrect.

End of solution.

Lecture 18

A *Boolean circuit* is a directed acyclic graph where each nonsource vertex, called a *gate*, is labeled with one of $\wedge$, $\vee$, or $\neg$. The inputs and outputs of a circuit are the source and sink vertices, respectively. The *size* of a circuit is the number of vertices in the graph.

If $x \in \{0,1\}^n$ and C is a circuit with n inputs and m outputs, then the output of C on x, written $C(x)$, is defined as follows. Let every input vertex v_i have the value x_i, and let every vertex labeled with a logical operator have the value got by applying that operator to the values of its incoming vertices. Then $C(x)$ is given by the values of the sink vertices. If $f\colon \{0,1\}^n \to \{0,1\}^m$ is a function, then we say that

$$C \text{ computes } f \iff C(x) = f(x) \text{ for all } x \in \{0,1\}^n$$

Claim 46. Every function $f\colon \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size $O(2^n)$.

Proof. Let $f\colon \{0,1\}^n \to \{0,1\}$ be any function. Note that

$$f(x_1,\ldots,x_n) = (x_n \wedge f(x_1,\ldots,x_{n-1},1)) \vee (\neg x_n \wedge f(x_1,\ldots,x_{n-1},0))$$

We can construct a circuit for f by simply evaluating this recursive equation and turning $\wedge$ and $\vee$ operators into gates. The number of $\wedge$ and $\vee$ gates for $f(x_1,\ldots,x_n)$ is given by the recurrence $G(n) = 2G(n-1)+3$, where $G(0)=0$, which means $G(n) = 3(2^n-1)$. We add 2^n inputs for the values of f and n gates connecting each x_i to $\neg x_i$. Thus for every function $f\colon \{0,1\}^n \to \{0,1\}$ we can construct a circuit of size $O(2^n)$ that computes it. $\square$

Exercise 93. Prove $\exists f\colon \{0,1\}^n \to \{0,1\}$ requiring circuits of size $2^{\Omega(n)}$.

Solution: Recall that there are 2^{2^n} functions from $\{0,1\}^n$ to $\{0,1\}$. We can show that there are $2^{O(k \lg k)}$ circuits of size k in the following way. Encode each circuit as a string over $\{0,1\}$ by using $4 \lg k$ bits to identify each gate: its (up to two) inputs plus its type (e.g., $\wedge$, $\vee$, $\neg$). Since there are $2^{4k \lg k}$ possible strings of length $4k \lg k$, there aren't more than that many circuits of size k. Let $k = 2^n/(5n)$. Then the number of circuits of size k is at most

$$2^{4k \lg k} \leq 2^{2^n 4n/5n} = 2^{2^n 4/5} < 2^{2^n}$$

That is, there is some function that requires circuits larger than $2^n/(5n)$. But note that $2^n/(5n) \geq 2^{cn}$ for some $c > 0$ and sufficiently large n, for example, for $c = 1/100$ and $n > 5$. Therefore $2^n/(5n) \geq 2^{\Omega(n)}$, and some function requires circuits of size $2^{\Omega(n)}$.

End of solution.

Exercise 94. Exhibit a function $f\colon \{0,1\}^* \to \{0,1\}$ that is not decidable but has circuits of polynomial size.

Solution: Consider the undecidable function that determines whether its input is in the language $\{1^n \mid n$'s binary expansion encodes a pair $\langle M, x\rangle$ such that M halts on input $x\}$. We describe a circuit family of linear size for this function. Define the circuit for inputs of length n to be a tree of $\wedge$-gates that computes $\wedge$ over all input bits if 1^n is in the language, and the circuit that always outputs 0 otherwise.

End of solution.

Lecture 19

19.42 Summary

Today we completed the circuits slides and started on the space slides. We got through the first 10 slides, finishing on the Space Complexity Theorem.

19.43 Exercises

Definition 47. Circuit-SAT $= \{C : C \text{ is a circuit and } \exists y : C(y) = 1\}$

Exercise 95. Reduce Circuit-SAT to 3SAT.

Solution: Assign a variable to each wire in the circuit and replace each gate in the circuit with 3-SAT clauses which guarantee the logic operation of that gate.

1. For NOT gate: if wire x is the input and wire y is the output then use these clauses to guarantee that $x = \neg y$:
   $(x \vee y \vee y) \wedge (\neg x \vee \neg y \vee \neg y)$.

2. For OR gate: if x and y are inputs and z is the output use the following clauses to ensure that $z = x \vee y$:
   $(z \vee \neg x \vee y) \wedge (z \vee x \vee \neg y) \wedge (\neg z \vee x \vee y) \wedge (z \vee \neg x \vee \neg y)$.

3. For AND gate: if x and y are inputs and z is the output use the following clauses to ensure that $z = x \wedge y$:
   $(\neg z \vee x \vee y) \wedge (\neg z \vee \neg x \vee y) \wedge (\neg z \vee x \vee \neg y) \wedge (z \vee \neg x \vee \neg y)$

To make sure that the final output of the circuit is also TRUE we should also add the clause $(w \vee w \vee w)$ in which w is the output wire. Finally combine all clauses with $\wedge$ operation.
$\Rightarrow$ If we have a TRUE assignment for the 3SAT problem then it also satisfies all gate operations in the circuit and also guarantees that the output is true.
$\Leftarrow$ If the Circuit-SAT problem has a satisfying assignment then the output is TRUE therefore all clauses in 3SAT problem will have at least one TRUE variable and 3-SAT problem will be satisfied too.
The process of constructing the 3SAT problem takes polynomial time in size of the circuit.
End of solution.

Exercise 96. 1. Prove $\exists c, \forall k, \Sigma_c P$ does not have circuits of size n^k.

2. Prove $PH \subseteq EXP$.

Solution:

1. First we show that there are languages which can't be accepted by circuit of size n^k:
For each gate we have have 3 choices which results in 3^{n^k} choices and also we have n^{2k} choices for connecting all gates. Therefore we have total of $3^{n^k}n^{2k}$ circuits of size n^k.
But since there are 2^{2^n} possible languages with input size n, there must be languages which can't be described by circuits of size n^k .
Now we use quantifiers to find a unique circuit with size bigger than n^k which accepts languages which can't be accepted by circuits of size n^k.

$x \in L_k \Leftrightarrow \exists C_1$ circuit of size n^{2k}
$\forall C_2$ circuits of size n^k
$\exists w$ input of size n
$C_1(w) \neq C_2(w)$
$\forall C_3$ circuits of size n^k, $C_3 \neq C_1$
$\exists C_4$ circuit of size n^k
$\forall w$ inputs of size n
$C_3(w) = C_4(w)$
$C_1(w) = TRUE$

Consider the machine M which runs the circuit with inputs of size n and does comparisons. M runs in polynomial time in size of its inputs. Therefore our language with 4 quantifiers is in $\sum_4 P$.

2. The proof is similar to the proof of $NP \subseteq EXP$.
Suppose $L \in \sum_i P$.
$\sum_i P = \{L : \exists$ poly-time M, polynomial q(n) :
$x \in L \Leftrightarrow \exists y_1 \in \{0,1\}^{q(n)} \forall y_2 \in \{0,1\}^{q(n)} \ldots Q y_{i+1} \in \{0,1\}^{q(n)} M(x, y_1, \ldots, y_{i+1}) = 1\}$

Let TM M' := "On input w,
for all tuples $\{\langle y_1, y_2, \ldots, y_{i+1}\rangle : |y_k| \leq |w|^{q(n)}\}$, run $M(w, y_1, y_2, \ldots, y_{i+1})$
If for all tuples $\{\langle y_2, y_4, \ldots, y_{i+1}\rangle : |y_k| \leq |w|^{q(n)}$, k is even$\}$,
there exists a tuple $\{\langle y_1, y_3, \ldots, y_{i+1}\rangle : |y_k| \leq |w|^{q(n)}$, k is odd$\}$,
which M accepts $(w, y_1, y_2, \ldots, y_{i+1})$, then ACCEPT;
otherwise REJECT"

M' accepts w$\leftrightarrow \exists y_1, \forall y_2, \exists y_3, \ldots, |y_i| \leq |w|^{q(n)}$, M accepts $(w, y_1, y_2, \ldots)$.
M' runs in time $2^{|w|^{q(n)^{i+1}}}|(w, y1, y2, \ldots, y_{i+1})|^{q(n)} \leq 2^{|w|^{q(n)^{i+1}+1}}$ which is exponential therefore $L \in EXP$
We can use similar proof for $L \in \prod_i P$.

End of solution.

Exercise 97. Prove $E \subseteq P/POLY \iff EXP \subseteq P/POLY$. Hint: use the padding technique.

Solution:
$\Leftarrow$ Since $E \subseteq EXP$ therefore if $EXP \subseteq P/POLY \rightarrow E \subseteq P/POLY$
$\Rightarrow$ consider $L \in EXP$ therefore there is a M which decides L in time 2^{n^c} for some c.
Consider language L'=$\{\langle x, 0^{n^c}\rangle : x \in L\}$
Define machine M' which first checks the input y to see if it is in form $\langle x, 0^{n^c}\rangle$. If it is not REJECT otherwise run M(x).
M' takes time $2^{|y|}$ so $L' \in E$ and therefore $L' \in P/Poly$
Now to find if input $x \in L$ we use the P/Poly circuit which decides L' to decide $\langle x, 0^{n^c}\rangle$
Therefore $L \in P/Poly$

End of solution.

Exercise 98. Write a definition of $NSPACE(\lg n)$ using pairs. Something like: $\exists y$ that makes the machine accept (x, y). Prove that $3SAT$ is in $NSPACE(\lg n)$. This should explain why we don't use that definition

Solution: We define $NSPACE(\lg n)$ as the set of languages such that $\exists M, y : L(M) \in SPACE(\lg n), x \in L \iff M$ accepts (x, y). It is easy to show that 3SAT is in this set. For a given problem x and satisfying assignment y, we can read the problem one clause at a time and check that it is satisfied by the assignment. This requires a constant amount of memory (to remember the variables in one clause), which is less than $\lg n$ for sufficiently large n.

End of solution.

Exercise 99. Show $TIME(t) \subseteq SPACE(t)$ for RATMs: the $TIME$ machine is random access, and the $SPACE$ machine is sequential. Hint: using technique similar to proof earlier. It's OK to use $O(1)$ work tapes.

Solution: We can simulate a RATM by using a normal TM with an extra work tape. This second tape should contain a list of (address, value) pairs for any location accessed by the RATM. We simply do a linear-time lookup when an address is read, adding a new entry with a blank value if we don't find it. Similarly, we do a linear-time lookup when an address is written, updating an existing entry with the new value.

The TM described can only touch one cell per operation, so if it runs in $t_{sim}(n)$ time it can only use $t_{sim}(n)$ space. It does take a polynomial factor more time than a RATM would take, because it needs to scan the second tape, but we can resolve this problem by observing that these extra operations do not contribute to the amount of space the TM is using. They are just spent searching for already-written data. Therefore, the space it consumes is at most $t(n)$. $\square$

End of solution.

Definition 48. The Space Hierarchy Theorem states that $\forall f, g : f(n) = o(g(n)), SPACE(f(n))$ is strictly contained in $SPACE(g(n))$.

Exercise 100. Prove the Space Hierarchy Theorem. Hint: You have to simulate a TM. You can use that the overhead in the simulation is a constant factor.

Solution: Let M_f be a Turing Machine running in space $f(n)$, and M_g be a Turing Machine running in space $g(n)$. By the same reasoning presented in the slides, M_f has at most $2^{O(f(n))}$ distinct configurations and M_g has at most $2^{O(g(n))}$.

If $f(n) = o(g(n))$ then there is some constant c for which $g(n) > cf(n)$ for any value of n. This can be used to show that M_g can have strictly more configurations than M_f:

$$2^{O(g(n))} > 2^{cO(f(n))} = (2^{O(f(n))})^c$$

Since M_g has more configurations, $SPACE(g(n))$ must strictly contain $SPACE(f(n))$. □

End of solution.

Lecture 20

20.44 Summary

In this lecture, we reviewed some previous exercises, and finished the discussion of space complexity.

20.45 Exercises

Exercise 101. Prove whether $\exists f\colon \{0,1\}^n \to \{0,1\}$ requiring circuits of size $2^{\Omega(n)}$.

Solution: Suppose such a function exists. Then $\exists n_0, \epsilon$ such that the size of the required circuit $s \geq 2^{\epsilon n}$ for all $n > n_0$. Consider a circuit of size s. Each gate in the circuit has an in-degree of at most 2. There are at most s^{3s} such circuits, which is the upper bound on the number of functions on n variables with circuits of size s. For $s = 2^{\epsilon n}$, this number is at most $2^{3\epsilon n 2^{\epsilon n}}$. For any $\epsilon \in (0,1)$, this number $< 2^{2^n}$, which is the total number of functions on n variables, for sufficiently large n. By pigeonhole principle, there must exist some function requiring circuits of size $\geq 2^{\epsilon n}$.

End of solution.

Definition 49. A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is computable in $SPACE(s(n))$ if the function $f'(x,i)\colon \{0,1\}^* \to \{0,1\}, f'(x,i) := f(x)_i$ is in $SPACE(s(n))$.

Exercise 102. Consider the alternative definition where TMs are equipped with a write-only tape that does not count towards space, where the TM is supposed to write $f(x)$. Show the two definitions are equivalent when, say, $|f(x)|$=poly$|x|$, $s(n) = O(\lg n)$.

Solution:
$\Rightarrow$
If $f'(x,i)\colon \{0,1\}^* \to \{0,1\}, f'(x,i) := f(x)_i$ is in $SPACE(s(n))$, then $\exists$ a TM M which computes f', then on input x, M can compute $f(x)$ bit by bit, using $f'(x,i) := f(x)_i$. Every time it halts, it writes the result on the write-only tape, and moves the head one cell right. Since the output tape does not count towards space, the total space needed is $s(n)$.
$\Leftarrow$
If TM N writes $f(x)$, then it works as a normal TM, while at some states, it writes $f(x)_i$ on the output tape, and moves the head of output tape one cell right. We can define M for $f'(x,i)$ such that M takes (x,i) as input, and halts when $f(x)_i$ is produced, that is, $f'(x,i) = f(x)_i$. It is obvious that $f'(x,i)$ is in $SPACE(s(n))$.

End of solution.

Exercise 103. Prove that multiplication can be computed in L.

Solution: We can use the same idea of grade-school multiplication. Suppose the two operands are a and b, m indicates the result, i, j, k is the bit index from right to left, then

we have:
$m_i = \sum_{j+k=i} a_j b_k + c$, where c is the carry bit from the previous column. Its initial value is 0. j, k take $O(\lg n)$ space, while c takes constant space, so total space needed is $O(\lg n)$.

End of solution.

Exercise 104. Prove that for any language A, $A \in NL \iff \overline{A} \in NL$.

Solution: Let M be the non-deterministic TM for A, $A = L(M)$. Let G be the computation graph of M on input w. We have:

$$w \in A \iff G \in PATH$$

If $|w| = n$, then $|G| = 2^{O(\lg n)}$. Testing if $G \in \overline{PATH}$ will take space

$$\lg |G| = \lg 2^{O(\lg n)} = O(\lg n)$$

End of solution.

Exercise 105. Prove that QBF is in PSPACE.

Solution: Consider the following algorithm:

Algorithm 2 A QBF Solver.

Solve-QBF$(Q_1 x_1 Q_2 x_2 ... Q_n x_n \varphi(x_1, x_2, ..., x_n))$
  if φ has no quantifier Q bounded **then**
    return the evaluation of $\varphi(x_1, x_2, ..., x_n)$
  end if
  if Q_1 is $\exists$ **then**
    return **Solve-QBF**$(Q_2 x_2 ... Q_n x_n \varphi(0, x_2, ..., x_n))$
      $\bigcup$ **Solve-QBF**$(Q_2 x_2 ... Q_n x_n \varphi(1, x_2, ..., x_n))$
  end if
  if Q_1 is $\forall$ **then**
    return **Solve-QBF**$(Q_2 x_2 ... Q_n x_n \varphi(0, x_2, ..., x_n))$
      $\bigcap$ **Solve-QBF**$(Q_2 x_2 ... Q_n x_n \varphi(1, x_2, ..., x_n))$
  end if

Because each recursive **Solve-QBF** call consumes a quantifier and there are at most n quantifiers, the space is bounded by $O(n^2)$.

End of solution.

Exercise 106. Explain why the $\forall$ quantifier is needed in the proof that QBF is PSPACE-hard

Solution: The $\forall$ quantifier appears in the following formula:

$$\varphi(c, c')_t = \exists d : \forall (a, b) \in \{(c, d), (d, c')\} : \varphi(a, b)_{t/2}$$

It allows us to reuse the φ expression at the end for both (c, d) and (d, c'). If we didn't do this, then each recursive step would double the amount of space we need to use for the QBF expression we are writing.

End of solution.

Lecture 21

21.46 Summary

In this lecture, we learned Randomized Complexity system.

21.47 Exercises

Exercise 107. (1)For RP, can replace $1/2$ with $1/n^c$, or $1-1/2^m$ for $m=n^c$, for any c.
(2)For BPP, can replace $(2/3, 1/3)$ with $(1/2+1/n^c, 1/2-1/n^c)$ or $(1-1/2^m, 1/2^m)$

Solution:
(1) $\frac{1}{2} \Rightarrow 1-\frac{1}{2^m}$:
If $L \in RP$. Define M':
Run M n^c times independently. If $M(x,R)=1$ for at least once, accept. Otherwise, reject.
Clearly, M' runs in polynomial time. If $x \notin L$, M will always reject x, so M' will always reject x. If $x \in L$, $Pr_R[M'(x,R)=1] = 1-(1-\frac{1}{2})^{n^c} = 1-(\frac{1}{2})^{n^c}$.
$\frac{1}{n^c} \Rightarrow \frac{1}{2}$:
If $\exists$ poly-time randomized M:
$x \in L \Rightarrow Pr_R[M(x,R)=1] \geq \frac{1}{n^c}$
$x \notin L \Rightarrow Pr_R[M(x,R)=1] = 0$
We can define poly-time randomized M':
Run M n^c times independently. If $M(x,R)=1$ for at least once, accept. Otherwise, reject.
Clearly, M' runs in polynomial time. And if $x \notin L$, M will always reject x, so M' will always reject x. If $x \in L$, $Pr_R[M'(x,R)=1] = 1-(1-\frac{1}{n^c})^{n^c} \geq 1-\frac{1}{e} \geq \frac{1}{2}$.
(2) (2/3,1/3)$\Rightarrow (1-1/2^m, 1/2^m)$:
If $L \in BPP$, define M':
Run M $k = \lceil 48m \ln 2 \rceil$ times. If M accepts x more than $k/2$ times, accept. Otherwise, reject.
Clearly, M' runs in polynomial time. We show the error rate of $M' \leq 1/2^m$.
Define $x_i = 1$ if in the i-th time M made the correct judgement; $x_i = 0$ if in the i-th time M made the wrong judgement, $i = 1, 2, ..., k$. $E(x_i) = p \geq 2/3$. By Chernoff Bound, the error rate of M' is

$$Pr[\sum_{i=1}^{k} x_i < k/2] \leq Pr[\sum_{i=1}^{k} x_i < (1-\frac{1}{4})\frac{2}{3}k] \leq \frac{1}{e^{\frac{1}{48}k}} \leq \frac{1}{2^m}.$$

$(1/2+1/n^c, 1/2-1/n^c) \Rightarrow (2/3, 1/3)$:

If $\exists$ poly-time randomized M:
$x \in L \Rightarrow Pr_R[M(x,R)=1] \geq 1/2 + 1/n^c$
$x \notin L \Rightarrow Pr_R[M(x,R)=1] \leq 1/2 - 1/n^c$
Define M':
Run M $k = \lceil \ln 3(2+n^c)n^c \rceil$ times. If M accepts x more than $k/2$ times, accept. Otherwise, reject.

Clearly, M' runs in polynomial time. We show the error rate of $M' \leq 1/3$.

Define $x_i = 1$ if in the i-th time M made the correct judgement; $x_i = 0$ if in the i-th time M made the wrong judgement, $i = 1, 2, ..., k$. $E(x_i) = p \geq 1/2 + 1/n^c$. By Chernoff Bound, the error rate of M' is

$$\begin{aligned} Pr[\sum_{i=1}^{k} x_i < k/2] &\leq Pr[\sum_{i=1}^{k} x_i < (1 - \frac{2}{2+n^c})(\frac{1}{2} + \frac{1}{n^c})k] \\ &\leq \exp(-\frac{1}{2}(\frac{2}{2+n^c})^2(\frac{1}{2} + \frac{1}{n^c})k) \\ &\leq \frac{1}{3}. \end{aligned}$$

End of solution.

Exercise 108. Show that the following are equivalent.

(1) L $\in$ RP $\cap$ co-RP

(2) There is a randomized poly-time machine M for L :
$\forall$ x, $\forall$ R, M(x,R)$\in$ {L(x), ?}
$\forall$ x, Pr_R [M(x,R) = ?] $\leqq$ 1/2

(3) There is a randomized machine M for L :
$\forall$ x, $\forall$ R, M(x,R) = L(x)
the expected running time of M on x is poly(n)

Solution:

1. (1) $\Rightarrow$ (2).

   From (1), $\exists$ M_{RP}, M_{co-RP}.

   Create $M_?$ on input x that uses M_{RP} and M_{co-RP} and operates as follows:

   - if $M_{RP}(x)$=1, emit 1
   - else if $M_{co-RP}(x)$=0, emit 0
   - else emit ?

   Claim 50. $M_?$ satisfies (2)

Proof. Case 1. Input x ∈ L.

Then, $M_?$ ∈ {1, ?} since M_{co-RP} never rejects x from the definition of co-RP. M_{RP} will accept x with probability $\geqq 1/2$ from definition of RP. Therefore, Pr_R [M(x,R) = ?] $\leqq 1/2$.

Case 2. Input x ∉ L. Vice Versa. □

2. (2) ⇒ (1).

   From (2), ∃ $M_?$.

   Claim 51. L ∈ RP

   Proof. Create M_{RP} that uses $M_?$ and emits 0 when $M_?$ emits ?. If input x ∈ L, M_{RP} rejects x with probability less than 1/2 from the definition of $M_?$. If input x ∉ L, M_{RP} always rejects. □

   Claim 52. L ∈ co-RP

   Proof. Vice versa of above proof □

3. (2) ⇒ (3).

   From (2), ∃ $M_?$.

   Create M_{LV} on input x that uses $M_?$ and operates as follows:

   - if $M_?(x)$=1, emit 1
   - else if $M_?(x)$=0, emit 0
   - else run $M_?(x)$ again recursively.

   Claim 53. M_{LV} always gives correct result.

   Proof. Trivial because $M_?$ always returns correct result unless the result is ? and M_{LV} never stops unless $M_?$ returns non-? result. □

   Claim 54. M_{LV} expects to finish in poly(n) time.

   Proof. Let one run of $M_?$ takes q(n) time which is poly(n) time. Pr_R [M(x,R) = ?] $\leqq 1/2$ from the definition of $M_?$.

   Therefore, expected runtime $\leqq \Sigma_{i=0}^{\infty} \frac{q(n)}{2^i} = 2 * q(n) = poly(n)$ □

4. (3) ⇒ (2).

   From (3), ∃ M_{LV}.

   Create $M_?$ on input x that uses M_{LV} and operates as follows:

   Run M_{LV} for 2*μ time where μ is the expected run time of M_{LV}. Return ? if M_{LV} does not finish until then.

Claim 55. $M_?$ returns ? with probability less than 1/2.

Proof. Since expected runtime of M_{LV} is μ, the probability that M_{LV} will not finish by $2^*\mu$ time is $\leqq 1/2$ by Markov's inequality. □

End of solution.

Lecture 22

22.48 Summary

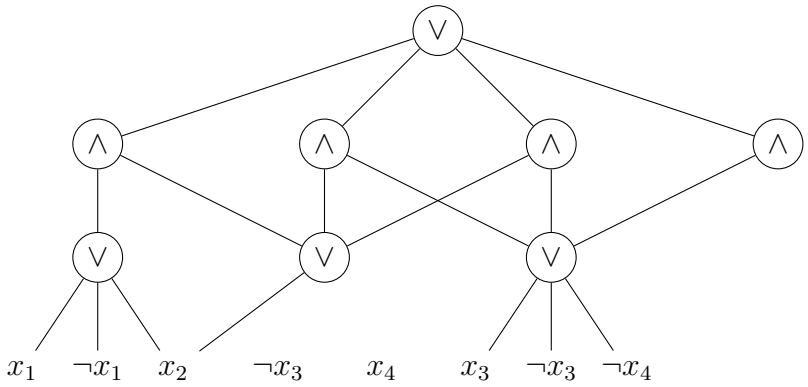
In this lecture, we mainly discuss AC^0 circuits and several claims about it.

22.49 Exercises

Definition 56. AC^0 circuits is defined as follows:

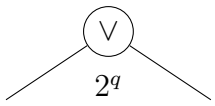
- unbounded FAN-IN,
- depth is a constant d (independent of n),
- gates $\vee$, $\wedge$ alternate on each level,
- input is x, $\neg x$.

One example of AC^0 circuit is illustrated as the following figure:



Depth-2 $AC^0 \equiv$ CNF, DNF.
PH can be seen as AC^0 on exponentially length inputs:

$$\exists\, y \in \{0,1\}^q \rightsquigarrow$$



In the proof that BPP $\subseteq$ PH. The exponentially length input is

$$M(x,\cdot) = \boxed{}\boxed{i}\boxed{}$$

$$M(x,i) \in \{0,1\}$$

The circuit is solving:

Definition 57. Approximate-Majority($\frac{1}{3}$,$\frac{2}{3}$), abbr., AMJ($\frac{1}{3}$,$\frac{2}{3}$), means that given a string z,

- does it have $\geq \frac{2}{3}|z|$ ones, or
- does it have $\leq \frac{1}{3}|z|$ ones.

otherwise, do not care.

Claim 58. AMJ($\frac{1}{3}$,$\frac{2}{3}$) has AC^0 circuits

- size is polynomial (in input length = $|z|$),
- depth = 3.

Proof. We will use probabilistic method. We show a distribution on circuits,denoted as C:

- $\forall\ z, |z| = n$ with $\geq \frac{2}{3}|z|$ ones, $Pr_C[C(z) = 0] < 2^{-n}$,
- $\forall\ z, |z| = n$ with $\leq \frac{1}{3}|z|$ ones, $Pr_C[C(z) = 1] < 2^{-n}$.

This $\Rightarrow Pr_C[\exists\ z : C(z) \neq \text{AMJ}(z)] < 2^n \cdot 2^{-n} < 1$.

Define:

- $\text{AMJ}_Y := \{z :\geq \frac{2}{3}|z| \text{ ones}\}$,
- $\text{AMJ}_N := \{z :\leq \frac{1}{3}|z| \text{ ones}\}$.

Pick AND of FAN-IN $c\lg_2(n)$, each input uniformly and independently on z:

$$\forall\ z \in \text{AMJ}_Y, Pr_C[C(z) = 1] \geq (\frac{2}{3})^{c\lg_2(n)} = n^{c\lg_2\frac{2}{3}} = \frac{1}{n^a}\ (a = -c\lg_2\frac{2}{3}) \tag{6}$$

$$\forall\ z \in \text{AMJ}_N, Pr_C[C(z) = 1] \leq (\frac{1}{3})^{c\lg_2(n)} = n^{c\lg_2\frac{1}{3}} = \frac{1}{n^b}\ (b = -c\lg_2\frac{1}{3}) \tag{7}$$

let $a < b$ by picking c large enough.
Negate AND,

$$\forall\ z \in \text{AMJ}_Y, Pr_C[C(z) = 1] \leq 1 - \frac{1}{n^a} \tag{8}$$

$$\forall\ z \in \text{AMJ}_N, Pr_C[C(z) = 1] \geq 1 - \frac{1}{n^b} \tag{9}$$

Take $n^{\frac{a+b}{2}}$ I.I.D copies of previous circuits, and then

$$\begin{aligned}
\text{For(3)}, Pr_C[C(z) = 1] &\leq (1 - \frac{1}{n^a})^{n^{\frac{a+b}{2}}} \\
&\leq e^{-n^{\frac{a+b-2a}{2}}} \\
&= e^{-n^{\frac{b-a}{2}}} \\
\text{Take a negate: } Pr_C[C(z) = 1] &\geq 1 - e^{-n^{\frac{b-a}{2}}}
\end{aligned}$$

$$\begin{aligned}
\text{For}(4), Pr_C[C(z)=1] &\geq (1-\frac{1}{n^b})^{n^{\frac{a-b}{2}}} \\
&\approx e^{-n^{\frac{a+b-2b}{2}}} \\
&= e^{-n^{\frac{a-b}{2}}} \\
&\approx 1-\frac{1}{n^{\frac{a-b}{2}}} \\
\text{Take a negate: } Pr_C[C(z)=1] &\leq \frac{1}{n^{\frac{a-b}{2}}}
\end{aligned}$$

Take n I.I.D copies, and then

$$\begin{aligned}
\text{For}(3), Pr_C[C(z)=1] &\geq (1-e^{-n^{\frac{b-a}{2}}})^n \\
&> 1-2^{-n}
\end{aligned}$$

$$\begin{aligned}
\text{For}(4), Pr_C[C(z)=1] &\leq (\frac{1}{n^{\frac{a-b}{2}}})^n \\
&< 2^{-n}
\end{aligned}$$

□

Comments of above proof: we just proved the existence, not an explicit construction. By contrast, BPP $\subseteq$ PH proof gives an explicit construction.

Exercise 109. Find out what circuits come out of BPP $\subseteq$ PH. Proof:

- size,
- depth,
- AMJ(α,β) for what α, β is solved?

Solution: Since M(x,R)=1 $\Leftrightarrow \exists s_1, ..., s_r : \forall y \in \{0,1\}^r, \vee_{i=1}^r M(x, y+s_i) = 1$, we can construct a AC^0 circuit that takes input z with length 2^r (fix x, each bit is M(x,R) for a coin toss R).
(1) The first layer (bottom layer) is a layer of $\vee$-gates, each corresponds to $\vee_{i=1}^r M(x, y+s_i) = 1$ for some y and some set $\{s_1, s_2, ..., s_r\}$. Every $\vee$-gate takes r bits of the input (FAN-IN r), and these r bits are $\text{M}(x, y+s_i)$ where $i = 1, 2, ..., r$.
(2) The second layer has $\wedge$-gates, each corresponds to some set $\{s_1, s_2, ..., s_r\}$. Every $\wedge$-gate chooses 2^r $\vee$-gates from the first layer, and these 2^r $\vee$-gates are $\vee_{i=1}^r \text{M}(x, y+s_i)$ for the same set $\{s_1, s_2, ..., s_r\}$ but different values of y. So each $\wedge$-gate has FAN-IN 2^r, because $y \in \{0,1\}^r$ has 2^r different values. The number of $\wedge$-gates is $O(2^{r^2})$ because there're $O(2^{r^2})$ sets like $\{s_1, s_2, ..., s_r\}$.
(3) The root is a $\vee$-gate that computes OR of the output from all the second layer's $\wedge$-gates.

So the size of this AC^0 circuit is $1 + O(2^{r^2}) + O(2^{r^2}) \cdot 2^r = O(2^{r^2} \cdot 2^r) = O(n^{1+\lg n})$, where $n = 2^r$.
The depth is 3.
AMJ($1/r^2$,$1 - 1/r^2$) is solved.
Name the AC^0 circuit C. As stated in the BPP$\subseteq$PH proof and also according to the construction of C, $C(z) = 1 \Leftrightarrow x \in \mathrm{L} \Rightarrow \Pr_R[\mathrm{M}(x, R)=1] \geq 1 - 1/r^2 \Leftrightarrow z \in \mathrm{AMJ}_Y$ where $\beta = 1 - 1/r^2$. On the other hand, $C(z) = 0 \Leftrightarrow x \notin \mathrm{L} \Rightarrow \Pr_R[\mathrm{M}(x, R)=1] \leq 1/r^2 \Leftrightarrow z \in \mathrm{AMJ}_N$ where $\alpha = 1/r^2$. So, when $z \in \mathrm{AMJ}_Y$, it must be $C(z) = 1$, otherwise if $C(z) = 0$ then $z \in \mathrm{AMJ}_N$ which is a contradiction. Similarly, when $z \in \mathrm{AMJ}_N$, it must be $C(z) = 1$. Therefore, C solves AMJ($1/r^2$,$1 - 1/r^2$).

End of solution.

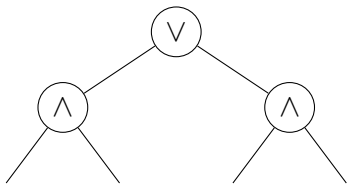
Claim 59. Depth-1 circuits cannot solve AMJ.

Proof. Say circuit is AND (if OR, similar). Suppose it touches x_1, then set $x_1 = 0$, rest to 1. Therefore, we have $\mathrm{AND}(x) = 0, x \in \mathrm{AMJ}_Y$ □

Claim 60. Depth-2 circuits of size Poly(n) (even $2^{o(n)}$) cannot solve AMJ.

Proof. Say circuit is

DNF =



Say $\exists$ some $\wedge$ with FAN-IN $\leq \frac{n}{3}$, set those to 1, rest to 0. DNF = 1, but input in AMJ_N.
Otherwise, all $\wedge$ of FAN-IN $\geq \frac{n}{3}$.
Let each bit x_i have $Pr[x_i = 1] = 0.9$

$$Pr[x \in \mathrm{AMJ}_Y] \geq \frac{1}{2} \quad (\text{ See the below exercise, Chernoff Bound}) \tag{10}$$

$$Pr[\mathrm{DNF}(x) = 1] \leq Pr[\exists \wedge : \wedge(x) = 1] \leq |circuit| \cdot (0.9)^{\frac{n}{3}} \tag{11}$$

combine (5) and (6), we have

$$|circuit| \geq 2^{\Omega(n)}$$

□

Exercise 110. Prove $Pr[x \in \mathrm{AMJ}_Y] \geq \frac{1}{2}$ using Chernoff Bound.

Solution: Define r.v. X_i, and $X_i = 1$ when $x_i = 1$, $X_i = 0$ when $x_i = 0$. So as stated in the above proof, $\Pr[X_i = 1]$=0.9.
According to Chernoff Bound, $\Pr[\sum_{i=1}^{n} X_i \leq (0.9 - \epsilon) \cdot n] \leq 1/2^{\epsilon^2 \cdot n}$.
Let $\epsilon = 0.9 - \frac{2}{3}$, we get $\Pr[x \notin \text{AMJ}_Y] \leq 1/2^{\epsilon^2 \cdot n}$, this probability is close to 0 when n is sufficiently large.
Therefore, $\Pr[x \in \text{AMJ}_Y] \geq \frac{1}{2}$.

End of solution.

Exercise 111. define

$$\text{parity } (x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum x_i \text{ is odd} \\ 0 & \text{otherwise} \end{cases}$$

Prove parity requires DNF of size $2^{\Omega(n)}$.

State-of-the-art: $\forall d$, parity requires depth-d circuit of size $\geq 2^{\Omega(n^{\frac{1}{d-1}})}$, e.g.,

- DNF of size $2^{\Omega(n)}$,
- Depth-3 of size $2^{\Omega(\sqrt{n})}$

Solution:

We first show that any $\wedge$-gate in the corresponding circuit must have fan-in of exactly n. Suppose some $\wedge$-gate has fan-in less then n, which means the circuit output doesn't depend on some x_i. Fix an input that makes some $\wedge = 1$, then the circuit output is 1. If we flip the x_i on which this $\wedge$-gate doesn't depend, the circuit output will stay 1, but the parity actually flips. So each $\wedge$-gate has fan-in of exactly n.

Then for each $\wedge$-gate, there is only one input that will make it output 1. Since there are 2^{n-1} inputs that have parity of 1, we need no less than 2^{n-1} $\vee$-gates. Therefore, parity requires DNF of size $2^{\Omega(n)}$.

End of solution.

Exercise 112. Let $A \in L$, show: $\exists\ d : A$ has an AC^0 circuit of depth-d of size $2^{O(\sqrt{n})}$.
Hint: Savitch, but break up in $\approx \sqrt{n}$ configurations.

Solution: We will show how to construct an AC^0 circuit of depth-d of size $2^{O(\sqrt{n})}$ for $A \in L$.

Let c_S be the start configuration of L machine, c_A be accept configuration, we want to know if $\text{REACH}(c_S, c_A, t) = 1$ where $t = n^c$ and L machine uses SPACE $c \lg n$. We know that

$$\text{REACH}(c_S, c_A, t) \quad \Leftrightarrow \quad \exists c_1 = c_S, c_2, \ldots, c_{n^{1/3}} = c_A, \forall i < n^{\frac{1}{3}}, \text{REACH}(c_i, c_{i+1}, t/n^{\frac{1}{3}}).$$

Actually, in this proof we can break the path from c_S to c_A into m parts for all $m < \sqrt{n}$, here we just pick $m = n^{\frac{1}{3}}$.

Now, according to the above fact, we construct the top two layers of AC^0 circuit as follows:

(1) The root is a $\vee$-gate, whose fan-in is the number of sequences like $c_1, \ldots, c_{n^{1/3}}$. The input of this $\vee$-gate is a set of $\wedge$-gates, each corresponds to a specific sequence of $c_1, \ldots, c_{n^{1/3}}$.

(2) For each $\wedge$-gate, each input bit is REACH$(c_i, c_{i+1}, t/n^{1/3})$ for some $i \leq n^{1/3}$.

Repeat the above procedure R times, until we get to REACH$(c_i, c_{i+1}, t/(n^{1/3})^R \leq \sqrt{n})$ questions where $t = n^c$, thus $R \geq 3c - 3/2$. Since each time we can generate one layer of $\vee$-gates and one layer of $\wedge$-gates, the depth of the AC^0 circuit $d = 2R \geq 2 \times (3c - 3/2) = 6c - 3$, which is a constant.

Note that each $\exists$ quantifier is over $n^{\frac{1}{3}} \cdot O(log(n))$ bits, and each $\forall$ quantifier is over $O(log(n))$ bits. So each $\vee$-gate has fan-in $2^{n^{1/3} \cdot O(log(n))}$, and each $\wedge$-gate has fan-in $n^{O(1)}$. On the other hand, as Reach$(c_i, c_{i+1}, \sqrt{n})$ depends on $\leq \sqrt{n}$ bits of the input, it can be computed by a DNF of size $\leq 2^{O(\sqrt{n})}$. Therefore, the total circuit size is $\leq 2^{\sqrt{n}} \cdot (2^{n^{1/3} \cdot O(log(n))} \cdot n^{O(1)})^R = 2^{O(\sqrt{n})}$.

End of solution.

Lecture 23

23.50 Summary

We went through the slides "Lower bounds" and discussed the branching programs in class. First, we showed that SAT can not be solved by an algorithm that runs in space $O(\lg n)$ and uses time n^c for a constant $c > 1$. This is equivalent to the following theorem.

Theorem 61. NTIME(n) NOT IN TIME(n^c)$\cap$L

which is based on the following two lemmas:

Lemma 62. L$\subseteq \cup_a \sum_a$TIME(n)

Lemma 63. NTIME(n) $\subseteq$ TIME $(n^c) \rightarrow \sum_a$TIME(n) $\subseteq$TIME(n^d) for $d = c^a$

23.51 Proof of Lemma 62

Proof. Let c_s be the start configuration of L machine, c_a be the accept configuration. We check REACH(c_s, c_a, t) where $t = n^c$ and the L machine uses SPACE ($c \lg(n)$). Note:

$$\text{REACH}(c, c', t) \iff \exists c_1 = c, c_2, c_3, \ldots, c_{\sqrt{n}} = c', \forall i < \sqrt{n}, \text{REACH}(c_i, c_{i+1}, t/\sqrt{n})$$

Since each time we check the existence of $\sqrt{n}$ configurations, the quantifier $\exists$ is actually over $\sqrt{n}O(\lg n)$ bits, which is $\ll n$. We repeat this R times by adding more quantifiers to go through each configuration, like the recursion part in the proof of Savitch Theorem. Finally, we have $t/\sqrt{n}^R$ REACH($c_i, c_i + 1, t/\sqrt{n}^R$) questions. By choosing $R = O(1)$, we make $t/\sqrt{n}^R \leq \sqrt{n}$ which means each REACH question can be answered in time $\sqrt{n}$. $\square$

23.52 Exercises

Exercise 113. Prove Lemma63

Solution: We prove based on induction.

Proof. For $a = 1$, we have $\sum_1$ TIME(n) = NTIME(n), and $c = d$, trivial.
Suppose case i is true, we prove case $i + 1$. Let language $L \in \sum_{i+1}$. Then by definition,

- $x \in L \iff \exists y_1, \forall y_2, \ldots Q_{i+1} y_{i+1} M(x, y_1, y_2, \ldots, y_{i+1}) = 1$
- We make y_1 part of input, and define another language L'
  $(x, y_1) \in L' \iff \exists y_2, \forall y_3, \ldots, Q_i M(x, y_1, y_2, \ldots, y_{i+1}) = 1$
- Clearly, $L' \in \prod_i$TIME(n) $\iff$ $L' \in$ co$\sum_i$TIME(n)
- Based on induction, we have $L' \in$ TIME(n^d) $\iff$ $\exists$ TM M' s.t. $M' = 1 \iff$ $(x, y_1) \in L'$

- Now we have $x \in L \iff (x, y_1) \in L' \iff \exists y_1, M'(x, y_1) = 1$ This is exactly the definition of NP. Therefore, $L \in \text{NTIME}(n^d)$
- Based on basic the rule $\text{NTIME}(n) \subseteq \text{TIME}(n^c)$, we have $\text{NTIME}(n^d) \subseteq \text{TIME}((n^d)^c)$ where $d = c^i$ and $d \times c = c^{i+1} \rightarrow \sum_{i+1} \text{TIME}(n) \subseteq \text{TIME}(n^{c^{i+1}})$

□

End of solution.

Definition 64. A branching problem is a directed acyclic graph where all nodes are labeled by variables, except for two output nodes labeled 0 or 1. The nodes that are labeled by variables are called query nodes. Every query node has two outgoing edges, one labeled 0 and the other labeled 1. Both output nodes have no outgoing edges. One of the nodes in a branching problem is designated the start node. A layered branching problem of width w length t is a graph like this. Each layer has $\leq w$ nodes, has connections pointing only to the next layer and is labeled by 1 variable. For width $\geq n$, the best time bound are of the form $t \geq n * \lg^c(n)$ for some c . (i.e. there exists $f \in NP$ that requires $t \geq n * \lg^c(n)$)

Exercise 114. Prove that if f requires depth 3 circuits of size $\geq 2^{n^{1/2+\Omega(1)}}$ then f would also require width n branching problems of length $t \geq n^{1+\Omega(1)}$.

Solution: A branching program of width n and length t can be seen as a configuration graph. We would like to know if $\text{REACH}(c_S, c_A, t) = 1$ where c_S is the start node in branching program, c_A is the output node labeled 0 or 1. t is the length of branching program. We know that

$$\text{REACH}(c_S, c_A, t) \iff \exists c_1 = c_S, c_2, \ldots, c_{n^{1/2}} = c_A, \forall i < n^{\frac{1}{2}}, \text{REACH}(c_i, c_{i+1}, t/n^{\frac{1}{2}}).$$

We can construct a circuit based on the definition of REACH at the right hand.

1. The root is a $\vee$-gate, whose fan-in is the number of sequences like $c_1, \ldots, c_{n^{1/2}}$. The input of this $\vee$-gate is a set of $\wedge$-gates, each corresponds to a specific sequence of $c_1, \ldots, c_{n^{1/2}}$.
2. For each $\wedge$-gate, each input bit is $\text{REACH}(c_i, c_{i+1}, t/n^{1/2})$ for some $i \leq n^{1/2}$.
3. $\text{Reach}(c_i, c_{i+1}, n^{1/2})$ depends on $\leq n^{1/2}$ bits of the input, it can be computed by a DNF of size $\leq 2^{O(n^{1/2})}$

This is a circuit of depth 3. In order to keep the size of the circuit $\geq 2^{n^{1/2+\Omega(1)}}$, the step of $t/n^{1/2}$ should be less than $n^{1/2}$. So t should be $\geq n^{1+\Omega(1)}$. f would require width n branching problems of length $t \geq n^{1+\Omega(1)}$.

End of solution.

For small width w (say $w = O(i)$), then you get bounds of $t \geq \Omega(\frac{n^2}{log(n)})$

Definition 65. The element distinction function $f : \{0,1\}^n \to \{0,1\}$ is defined as new input $x \in \{0,1\}^n$ as $\frac{n}{2log(n)}$ values and $Y_i \in [n^2]$

Claim 66. $\forall w = O(1)$ f requires branching programs of length $\Omega(\frac{n^2}{log(n)})$

Proof. Consider a width w, length t branching program for f. Partition the layers according to the $\frac{n}{2log(n)}$ blocks of the input. Some set of the partition contains $\leq \frac{t}{n} * 2 * log(n)$ elements. Say this set corresponds to Y_{i^*} when we fix all $Y_j : j \neq i^*$. Thus, we obtain a branching program of length $\frac{t}{n} * 2log(n)$
The number of such branching programs is $\leq 2^{O(wlog(w)*\frac{t}{n}*2*log(n))}$.
Since we can consider $wlog(w)$ a constant, the number of branching programs is $\leq 2^{O(\frac{t}{n}log(n))}$
On the other hand, by fixing Y_j for $j = i^*$, you obtain $\geq 2^{\Omega(n)}$ different functions starting from f. And since $2^{\frac{t}{n}*2*log(n)} \geq 2^{\Omega(n)}$ then $t \geq \Omega(\frac{n^2}{log(n)})$.

Exercise 115. Show that you obtain $\geq 2^{\Omega(n)}$ different functions of Y_{i^*} by fixing $j \neq i^*$.

Solution: Since there are $\frac{n}{2log(n)}$ blocks of x and each block can have $2log(n)$ distinct values, if we fix $j \neq i*$ then there must be at least $2^{2log(n)^{\frac{n}{2log(n)}}}$ or $\geq 2^{\Omega(n)}$ functions.
End of solution.

Lecture 24

In this class, we started by reviewing previous exercises, some recent major results in complexity theory, and finished with the big picture of the relationships between different complexity classes.

24.53 Writing Proofs

- If the proof remains invariant under changing the question's parameters, it is flawed.

24.54 Achievements in Complexity Theory

- **PCP Theorem** (Probabilistically Checkable Proofs): It allows us to construct mathematical proofs in a form that can be checked by looking at few probabilistically chosen symbols (from the proof). One of the main applications of this theorem is in *inapproximability*.

- **Pseudorandomness**: TIME($2^{O(n)}$) requires circuits of SIZE($2^{\Omega(n)}$) $\implies P = BPP$. This theorem implies that if some functions are hard, then randomness is useless. A sufficiently hard function can give the illusion of randomness.

24.55 The Big Picture

The figure below summarizes the relationships between different complexity classes.

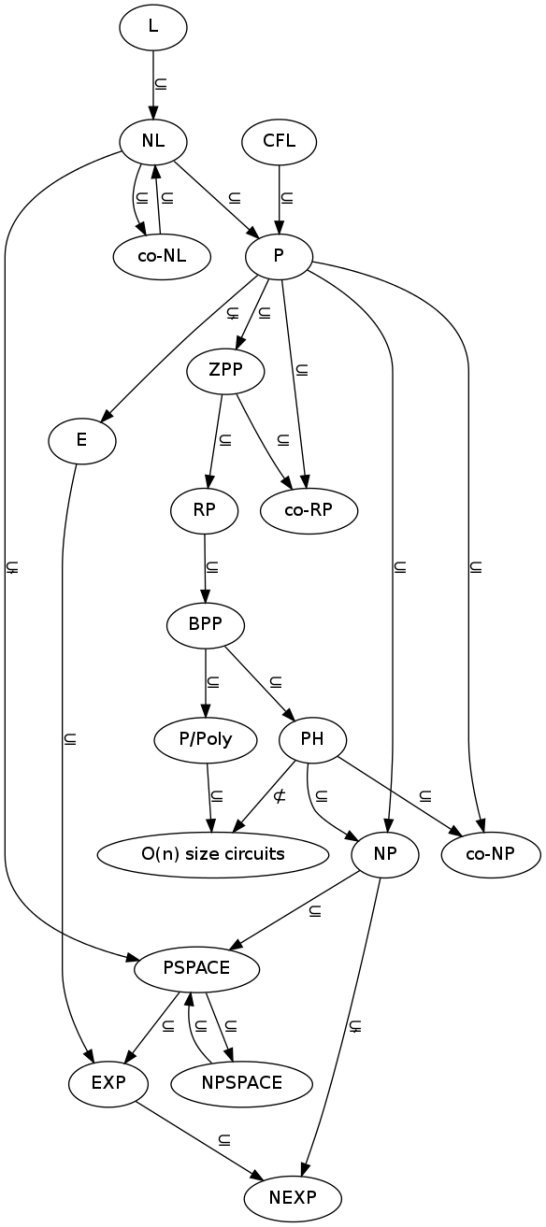


Figure 3: Relationships between different complexity classes