Computational Complexity

Nitin Saxena

Winter term 2008/09 - University of Bonn, Germany

Contents

Chapter 1

Formalizations

1.1 Formalizing Problems & Language

Definition 1.1.1. *The language generated by a finite set M is defined as*

$$M^* := \left\{ s \in M^k \mid k \in \mathbb{N} \right\}$$

Definition 1.1.2. *A problem is a function $f : \{0,1\}^* \mapsto \{0,1\}^*$*

Example 1.1.3. *addition of $a, b \in \mathbb{Z}$ can be done in*

$$O(\log(\text{sizeof}(a)) + \log(\text{sizeof}(b)))$$

Remark 1.1.4. *all objects like "integer x", "graph G", "vector v" ... will be finite and we can fix a dictionary that interprets all characters $\{a_1, a_2, \ldots\}$ as $\{1, 11, \ldots\}$ and uses 0 as a seperator.*

Definition 1.1.5. *A Problem f is called boolean or decision problem if $f : \{0,1\}^* \mapsto \{0,1\}$*

Usually problems can be reduced to such boolean problems. For example addition can be reduced to g: $(+, i) : \mathbb{Z} \times \mathbb{Z} \mapsto \{0,1\}$ s.t. $\forall a, b \in \mathbb{Z} : g(a,b)$ is the i-th bit of $(a+b)$. A problem f has an associated set denoted by

$$L_f := \left\{ x \in \{0,1\}^* \mid f(x) = 1 \right\}$$

this set is called language of f. So we have a strong connection between languages, decision problems and subsets of $\{0,1\}^*$.

Example 1.1.6. *"testing whether a number is prime or not" $\leftrightarrow f : \{0,1\}^* \mapsto \{0,1\} \leftrightarrow PRIMES := \left\{ x \in \{0,1\}^* \mid x \text{ is prime} \right\}$*

Definition 1.1.7. *Identify "decide a language L_f" and "compute a boolean f".*

1.2 Formalizing Machines

Definition 1.2.1. *A Turing machine M consists of*

- *a finite control-tape of (Γ, Q, δ)*

- *a possibly infinite data-tape*

such that

- Γ *is the alphabet of M, it always contains of* $\triangleright$ *(start),* $\square$ *(blank),* 0, 1 *and any other character, but finitely many*

- Q *is a finite set of stats (we say that* M *is in state ...) and it has two special states:* q_s *(starting state) and* q_f *(ending-state)*

- $\delta : Q \times \Gamma^2 \to Q \times \Gamma \times \{L, R, S\}^2$ *is a transition function of* M

A configuration of M *comprises of (state, input-head, work-head) and an application of* δ *changes the configuration. One step of* M *looks like: Say the current configuration is* (q, i, w) *and* $\delta(q, i, w) = (q', w', \epsilon_1, \epsilon_2)$ *than the action of* δ *means:*

- *change the state to* q'

- *write* w' *to the cell where the work-head is*

- *move the input head one cell in the direction* $\epsilon_1 \in \{L = left, R = right, S = stay\}$ *(if it is possible) and move the work-head in the direction* ϵ_2

Example 1.2.2. *abc*
$(\triangleright, \triangleright) \mapsto (\triangleright, R, S)$
$(0, \triangleright) \mapsto (\triangleright, R, S)$
$(1, \triangleright) \mapsto (\triangleright, R, S)$
$(\square, \triangleright) \mapsto (\triangleright, L, R)$
$(\triangleright, \square) \mapsto (\square, S, L)$
$(0, \square) \mapsto (1, L, L)$
$(1, \square) \mapsto (0, L, L)$

Remark 1.2.3.

- *The number of states* ($|\delta|$) *represents the size of a C-program*

- *The number of work-cells that* M *actually uses represent the working-space of a program that is used*

- *The number of* δ*-applications on the start configuration to reach the final configuration is the time used by a program when executed*

Exercise 1.2.4.

- *Simulate high-school methods of addition / multiplikation on a turing machine*

- *proof that any C-program has a corresponding turing machine and vice versa*

So a turing machine is just a “computer” with an infinite harddisk.

1.3 Church-Turing-Thesis

The Curch-Turing-Thesis says that every physically realizable computing dvice - silicon based, DNA-based or alien technology - can be simulated on a turing machine.

Definition 1.3.1. *We say a problem f is computable or decidable or rekursive if there exists a turing machine M s.t. $\forall x \in \{0,1\}^*$ it holds that M gives the output $f(x)$ in finite time. A boolean Funkton is called computable enumerable, recursively enumerable or decidable enumerable when there is a turing machine that "prints" L_f.*

After this Definition the questien comes up if there are uncomputable functions that are computable enumerable.

Definition 1.3.2.

$$HALT := \left\{(M,x) \mid M \text{ is a turing machine}, x \in \{0,1\}^*, M \text{ halts on } x\right\}$$

Theorem 1.3.3. *$HALT$ is enumerable but $\overline{HALT}$ is not.*

To proof theis theorem we will give turing machines as input to other turing machines. Every description of a turing machine ... **TODO:** go on

Definition 1.3.4. *Diophantine Problem: Given a $\mathbb{Z}$-polynomial, decide it's solvability over $\mathbb{N}$.*

- posed by Hilbert (1900)
- Turing studied the Halting Problem 1.3.2 (1936)
- a plan was suggested by Davis, Putnam & robinson (1950-1960)
- the proof was completed by Matiyasevich (1970)

We will basically proof a stronger theorem:

Theorem 1.3.5. *If R is a computably enumerable language then*

$$\exists P_R \in \mathbb{Z}[x, x_1, \dots, x_n]$$

s.t. $\forall b \in \{0,1\}^* \cap R$ the equation $P_R(b, x_1, \dots, x_n) = 0$ is solvable over $\mathbb{N}$. P_R is called a Diophantine representation.

Corrolar 1.3.6. *If $HALT$ has a Diophantine representation, the Diophantine Problem is uncomputable.*

Corrolar 1.3.7. *Even for polynomials of degree 4 its undecidable.*

Exercise 1.3.8. *Let K be a field and $P \in K[x_1, \dots, x_n]$. Show that the polynomial equation $P(x_1, \dots, x_n) = 0$*

a) has an equivalent quadratic system

b) can be reduced to an equation of degree ≤ 4

Corrolar 1.3.9. *$\exists f \in \mathbb{Z}[\overline{x}]$ s.t all its positive values on $\mathbb{N}$-evaluations is exactly $PRIMES$ i.e.:*

$$PRIMES = \{f(n) \mid n \in \mathbb{N} \ \& \ f(n) > 0\}$$

Proof. $PRIMES$ is computably enumerable. So by 1.3.5 a polynomial

$$P_{PRIMES}(x, x_1, \ldots, x_n)$$

exists. Define

$$f(x, x_1, \ldots, x_n) := x \cdot (1 - P^2_{PRIMES}(x, x_1, \ldots, x_n))$$

Now we see that (for $\overline{x} = (x_1, \ldots, x_n)$)

$$f(\overline{x}) > 0 \Leftrightarrow x > 0, (1 - P^2_{PRIMES}) > 0 \Leftrightarrow x > 0, P_{PRIMES} = 0$$

□

First step in the proof of 1.3.5: (representing a turing-machine computation by an arithmetic formular in $+, -, *, =, <, \ \& \ , \vee, \forall, \exists$)

Let R be enumerable by a turing-machine M_R. We will construct a first-order arithmetic formula $F_R(b)$ s.t. $b \in R \Leftrightarrow F_R(b) = true$. Example:

$$\forall x_1 \exists x_3 \forall x_3 \left[x_1^2 > x_2 + x_3 \ \& \ x_3 > 0\right]$$

F_R will be built by using:

atomic formulas $t_1 = t_2, t_1 < t_2$ where $t_1, t_2 \in \mathbb{N}[\overline{x}]$.

logical operators & (logical "and"), $\vee$ (logical "or")

existential quantifier $\exists$ (universe = $\mathbb{N}$)

restricted universal quantifier $\forall z < U$ (universe = $\mathbb{N}$ and $U \in \mathbb{N}$ fixed)

Proposition 1.3.10. *If a turing-machine M_R enumerates R then we can construct a first-order arithmetic formula $F_R(b)$ that imitates M_R step-by-step till it prints b only by using the above.*

Proof. We construct

- a first-order arithmetic formula $F_R(b)$ that imitates M_R step-by-step until it prints b.
- First attempt: Assume one head and one tape in M_R
  - associate a configuration of M_R with a vector

    $$C := \left[s(C), p(C), q(C), a_0(C), \ldots a_{q(C)-1}(C)\right]$$

    where s is the state, p is the number of the cell where the head points to, q is the number of used cells and a_i are the bits on the tape.
  - $F_R(b) := \exists C_1 \exists C_2 \left[Start(C_1) \ \& \ Compute(C_1, C_2) \ \& \ Stop(C_2, b)\right]$
  - $Start(C_1)$: asserts the start configuration

    $$(s(C_1) = q_{start}) \ \& \ (p(C_1) = 1) \ \& \ (a_0(C_1) = \square)$$

- $Stop(C_2, b)$: asserts that $b = (b_0, \dots b_{m-1})$ has been printed

$$(\exists m < q(C_2))\ \&\ (\forall i < m)(a_i(C_2) = b_i)\ \&\ (a_m(C_2) = \square)$$

- $Compute(C_1, C_2)$: asserts that there is a configuration sequence

$$L = [g_0(L), \dots, g_w(L)]$$

s.t. M_R follows them to reach C_2 from C_1

$$\exists L \exists w (g_0(L) = C_1)\ \&\ (g_w(L) = C_2)\ \&$$
$$((\forall i < w) \exists C_3 \exists C_4 (g_i(L) = C_3)\ \&\ (g_{i+1}(L) = C_4)$$
$$\&\ \bigwedge_{I \in \delta_{M_R}} Step_I(C_3, C_4))$$

- $Step_I(C_3, C_4)$: assert that C_3 to C_4 is a step following an instruction $I \in \delta_{M_R}$: Say, $I = (s, b) \mapsto (s', b', \epsilon); \epsilon \in \{L, R, S\}$. For $\epsilon = S$:

$$\begin{array}{rcccl} (s(C_3) = s) & \& & (s(C_4) = s') & \& & \\ (\exists k \exists n (& & (p(C_3) = k) & \& & (p(C_4) = k) \\ & \& & (a_k(C_3) = b) & \& & (a_k(C_4) = b') \\ \&\ (\forall i < n) & & (a_i(C_3) = k) & \vee & a_i(C_4)\quad)) \end{array}$$

Exercise 1.3.11. *Do it for $\epsilon = L$ and $\epsilon = R$*

- the problem is we are quantifying over vectors C_1 and C_2 of variable length (even worse: L is a vector of vectors).
- Remedy: Try encoding a vector $[a_0, \dots, a_{n-1}]$ as a pair (x, y) of integers.
- How to decode a_i back from (x, y) arithmetically?

□

Theorem 1.3.12. *Chinese Remainder Theorem: If m, n are coprime integers then*

$$\forall a, b\ \exists x\, [x \equiv a \pmod m\ \&\ x \equiv b \pmod m]$$

or equivalently: If $gcd(m, n) = 1$ then $\mathbb{Z}/mn\mathbb{Z} \cong \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$.

Exercise 1.3.13. *Proof 1.3.12*

- Encode $C = [n, a_0, \dots, a_{n-1}]$ as (x, y) s.t.

$$\begin{array}{rcll} n & \equiv & x & \pmod{y+1} \\ a_0 & \equiv & x & \pmod{2y+1} \\ a_1 & \equiv & x & \pmod{3y+1} \\ & \dots & & \\ a_{n-1} & \equiv & x & \pmod{ny+1} \end{array}$$

- Decoding requires modulo computation:

$$a \equiv b \pmod c \Leftrightarrow (\exists d)\,[(a = b + cd) \vee (a = b - cd)]$$

- There always exists (x, y) encoding $[a_0, \dots, a_{n-1}, n]$. Proof. it suffices to use a y s.t. $\{y+1, 2y+1, \dots, (n+1)y+1\}$ are mutually coprimte. Verify that $\mathrm{lcm}(1, \dots, n+1) \mid y$ then it works. □

- How to express $y \equiv 0 \pmod{\mathrm{lcm}(1, \dots, n+1)}$ arithmatically?

$$\mathrm{lcm} \mid y \Leftrightarrow (\forall i < n+2)(y \equiv 0 \pmod{i})$$

Exercise 1.3.14. *Finish the construction of $F_R(b)$.*

Proposition 1.3.15. *If a first-order arithmetic formula has no universal quantifier then we can derive a Diophantine instance from it.*

Proof. We distinct the following cases:

a) $(\exists z)(P(z) = 0)$ is just a Diophantine instance

b) $\exists z_1 \exists z_2 : (P(z_1) = 0 \ \& \ P(z_2) = 0) \Leftrightarrow \exists z_1 \exists z_2 : (P^2(z_1) + P^2(z_2) = 0)$

c) $\exists z_1 \exists z_2 (P(z_1) = 0 \vee Q(z_2) = 0) \Leftrightarrow \exists z_1 \exists z_2 (P(z_1) Q(z_2) = 0)$

d) $\exists z_1 \exists z_2 (P(z_1) < Q(z_2)) \Leftrightarrow \exists z_1 \exists z_2 \exists z (P(z_1) + z + 1 - Q(z_2) = 0)$

Induction finishes the proof. □

So lets proof 1.3.5 assuming we already have a Diophantine representation of:

Binomial Coefficient $x = {}^{y}_{z}$

Factorial $x = y!$

Exponential function $x = y^z$

Proof. Let M_R be the enumerator of R. We have already constructed F_R in 1.3 and the only part of it left to convert to a Diphantine problem is:

$$(\forall z < U) \exists x_1 \dots \exists x_n (P(b, z, x_1, \dots, x_n) = 0)$$

Our aim is to get an equivalent formula that looks like

$$\exists y_1, \dots \exists y_m (Q(b, y_1, \dots, y_m) = 0)$$

The assertion is actually existential, it says there exists:

$$\begin{array}{lrcl} \text{for} & z = 0 & : & x_1^{(0)}, \dots, x_n^0 \\ \text{for} & z = 1 & : & x_1^{(1)}, \dots, x_n^1 \\ \text{for} & z = U-1 & : & x_1^{(U-1)}, \dots, x_n^{U-1} \end{array}$$

We want to compress the columns $[0, \dots, U-1]$ as Z and $x_i^0, \dots x_i^{U-1}$ as w_i. So we use the Chinese Remainder idea again: suppose we fix U coprime integers $u_0, \dots, u_{U-1}$ then there is a vector $(U, w_1, \dots, w_n)$ s.t.:

$$\forall z \in \{0, \dots, U-1\} \forall i \in \{1, \dots, n\}, w_i \equiv x_i^{(z)} \pmod{u_z}$$

This encodes the z-th row by $(Z, w_1, \dots, w_n)$ but modulo u_z. Because now

$$P(b, Z, w_1, \dots, w_n) \equiv 0 \pmod{u_z} \quad \forall z < U$$

or equivalently

$$P(b, Z, w_1, \ldots, w_n) \equiv 0 \pmod{u_0 \cdot \ldots \cdot u_{U-1}}$$

So we have captured zeroness $\pmod{u_z}$ and we make it absolut by choosing u_z large enough. So its remains to find out how large we have to choose u_z. Say $|x_i^{(z)}| < X \quad \forall z$, $N = \deg(P)$ and M = sum of absolute coefficients in P. Then

$$|P(b, z, x_1, \ldots, x_n)| < M \cdot ((b+1) \cdot U \cdot X)^N =: T$$

Thus forcing $u_z > T$ gives us:

$$P(b, Z, w_1, \ldots, w_n) \equiv 0 \pmod{u_z} \Rightarrow P(b, z, x_1^{(z)}, \ldots, x_n^{(z)}) \pmod{u_z}$$

Since $u_z > T > |P(b, z, x_1^{(z)}, \ldots, x_n^{(z)})|$ it follows that $P(b, z, x_1^z, \ldots, x_n^{(z)}) = 0$.

So we need coprime u_zs whose product has a Diophantine representation. Assuming the representation of the binomial coefficient, let us fix:

$$\begin{array}{rcl} u_0 \cdot \ldots \cdot u_{U-1} & = & \binom{v}{u} \\ & = & \frac{v(v-1)\ldots(v-u+1)}{1 \cdot 2 \cdot \ldots \cdot u} \\ & = & \left(\frac{v+1}{1} - 1\right) \cdot \left(\frac{v+1}{3} - 1\right) \cdot \ldots \cdot \left(\frac{v+1}{U} - 1\right) \end{array}$$

Pick $u_z = \left(\frac{v+1}{z+1} - 1\right)$.

- To make $u_z \in \mathbb{N} : (u!)|(v+1)$
- Note that $gcd(\frac{v+1}{1} - 1, \frac{v+1}{j} - 1) = gcd(\frac{v+1}{i} - 1, \frac{v+1}{i}\frac{j-i}{j}) = gcd(\frac{v+1}{i} - 1, \frac{j-i}{j})$
- Show that $u_z > u_{z+1}$

Exercise 1.3.16. *a) If we force $(u!)^2|(v+1)$ then $u_0, \ldots, u_{U-1}$ are coprime naturals.*

b) $v \equiv z \pmod{u_z}$

G_1**:** $P(b, V, w_1, \ldots, w_n) \equiv 0 \pmod{\binom{v}{u}}$

$\forall i \in \{1, \ldots, n\} : G_{2,i} \;\; (\forall z < U)(w_i \pmod{\left(\frac{v+1}{z+1} - 1\right)} < X) :$

G_3**:** $\left(\frac{v+1}{U} - 1\right) \geq M \cdot ((b+1) \cdot U \cdot X)^N$

G_4**:** $v + 1 \equiv 0 \pmod{(u!)^2}$

Exercise 1.3.17. *Verify that*

$$\exists X \exists V \exists w_1, \ldots \exists w_n (G_1 \;\&\; G_{21} \;\&\; \ldots \;\&\; G_{2n} \;\&\; G_3 \;\&\; G_4)$$

is equivalent to

$$(\forall z < U) \exists x_1 \ldots \exists x_n (P(b, z, x_1, \ldots, x_n) = 0)$$

The only kind of universal quantifier that we need to remove is:

$$(\forall z < U)(w_i \pmod{u_z < X} \ldots w_i \pmod{u_z < X})$$

if and only if (u_z devides at least one of $w_i, w_i - 1, \ldots, w_i - X + 1$). The second statement implies that $u_z | w_i \cdot (w_i - 1) \cdot \ldots \cdot (w_i - X + 1)$ which is equivalent to $u_z | \frac{w_i!}{(w_i - X)!}$. So $(\forall z < U)(w_i \pmod{u_z < X} \Rightarrow \overset{v}{u} | \frac{w_i!}{(w_i-X)!}$. If we know how to express $\overset{v}{u}$ and factorial then we can express the condition

$G'_{2,i}$**:** $\overset{v}{u} | \frac{w_i!}{(w_i-X)!}$

which says that

$$\forall z \in \{0, \ldots, U-1\} \forall i \in \{1, \ldots, n\} : u_z | \frac{w_i!}{(w_i - X)!}$$

So Fix a $z \in \{0, \ldots, U-1\}$ and from the fact $u_z | \frac{w_i!}{(w_i-X)!}$ we get:

$$\exists S_1 | (u_z), S_1 > u_z^{\frac{1}{x}} \exists y_1^{(z)} < X, S_1 | (w_1 - y_1^{(z)})$$

from $S_1 | \frac{w_2!}{(w_2-X)!}$ we get:

$$\begin{array}{rcl} \exists S_2 | S_1, S_2 & > & u_z^{\frac{1}{x^2}} \exists y_2^{(z)} < X, S_2 | (w_2 - y_2^{(z)}) \\ & \cdots & \\ \exists S_n | S_{n-1}, S_n & > & u_z^{\frac{1}{x^n}} \exists y_n^{(z)} < X, S_n | (w_n - y_n^{(z)}) \\ \Rightarrow S_n & > & (u_z)^{\frac{1}{x^n}} \ \& \ S_n | (w_1 - y_i^{(z)}, \ldots, (w_n - y_n^{(z)}) \\ \Rightarrow P(b, V, w_1, \ldots, w_n) & \equiv & P(b, z, y_1^{(z)}, \ldots, y_n^{(z)}) \equiv 0 \pmod{S_n} \end{array}$$

So if we choose $S_n > T$ then $P(b, z, y_1^{(z)}, \ldots y_n^{(z)}) = 0$. So choose

G'_3**:** $u_{U-1} \geq (M((b+1)XU)^N)^{x^n}$

Exercise 1.3.18. *Verify that:*

$$\exists X \exists V \exists w_1 \ldots \exists w_n (G_1 \ \& \ G'_{21} \ \& \ \ldots \ \& \ G_{2n'} \ \& \ G'_3 \ \& \ G_4)$$

is equivalent to

$$(\forall z < U) \exists x_1 \ldots \exists x_n (P(b, z, x_1, \ldots x_n) = 0)$$

□

Lemma 1.3.19. *We can find a Diophantaine representation of the binomial coefficients using that of the exponential: That means, we want a Diophantaine representation of the predicat:*

$$(z \leq y) \ \& \ \overset{y}{z}:$$

Proof.

$$\begin{array}{rcl} (1+p)^y & = & \sum_{i=0}^{y} \overset{y}{i} \, p^i \\ & = & u + (\overset{y}{z} + pv) p^z \\ u & := & \sum_{i=0}^{z-1} \overset{y}{i} \, p^i \\ v & := & \sum_{i=z+1}^{y} \overset{y}{i} \, p^{i-z-1} \end{array}$$

So we can hope to

a) quotient$({}^{(1+p)^y}/_{p^3})$

b) and by $\pmod{p}$ we get $\binom{y}{z}$

The two divisions (first by p^3 and the second by p) give $\binom{y}{z}$ if and only if $u < p^z$ & $\binom{y}{z} < p$. So let us choose $p = (3^y + 1)$:

a)

$$\begin{array}{rcl} u & = & \sum_{i=0} z-1 \binom{y}{i} p^i \\ & < & 2^y \sum_{i=0} z-1 p^i \\ & = & 2^y \frac{p^z-1}{p-1} \\ & = & 2^y \frac{p^z-1}{3^y} \\ & < & p^z \end{array}$$

b) $\binom{y}{z} < 2^y < p$

The two divisions will extract $\binom{y}{z}$ from $(1+p)^y$ for $p = (3^y + 1)$.

Exercise 1.3.20. *Verify that:* $(z \le y)$ & $(x = \binom{y}{z})$ *is equivalent to*

$(z \le y)$ & $\exists p \exists u \exists v((p = 3^y + 1)$ & $((1+p)^y = u + (x + vp)p^z)$ & $(u < p^z)$ & $(x < p))$

So a Diophantaine representation of exponential yields a Diophantaine representation of $x = \binom{y}{z}$. □

Lemma 1.3.21. *We can find a Diophantaine representation of factorial:*

$$x = y!$$

using exponential and binomial coefficients.

Proof. Note that $y!$ appears in $\binom{w}{y} = \frac{w!}{y!(w-y)!}$. Let $w >> y$ then $\binom{w}{y} \approx \frac{w^y}{y!} \Rightarrow y! = \frac{w^y}{\binom{w}{y}}$. So how big should w be?

$$y! \le \frac{w^y}{\binom{w}{y}} = y! \frac{w}{w} \frac{w}{w-1} \cdots \frac{w}{w-y+1} \le y! \left(\frac{w}{w-y}\right)^y = y! \left(1 + \frac{y}{w-y}\right)^y$$

So let $\frac{1}{t} = \frac{y}{w-y}$:

$$\begin{array}{rcl} \frac{w^y}{\binom{w}{y}} & \le & y!(1+\frac{1}{t})^y \le y!(1+\sum_{i=1}^{y} \binom{y}{i} \frac{1}{t^i}) \\ & \le & y!(1+\frac{2^y y}{t}) = y! + \frac{2^y y y!}{t} \end{array}$$

let $t = 2^y y y^y$, then $\frac{w^y}{\binom{w}{y}} \le y! + \frac{1}{2}$ and note that $w = y + 2^{y+1} y y + 2$ and that $y! \le \frac{w^y}{\binom{w}{y}} \le y! + \frac{1}{2}$. By keeping the integral part of this term we have computed $y!$. □

Exercise 1.3.22. *Verify that:* $x = y!$ *is equivalent to*

$$\exists w \exists r \left((w = y + 2^{y+1} y^{y+2}) \ \& \ \left(w^y = x \binom{w}{y} + r\right) \ \& \ \left(r < \binom{w}{y}\right)\right)$$

Lemma 1.3.23. *We will now reduce exponential to a special exponential: Compute v^n modulo a small polynomial in v (e.g. $v^2-2av+1$). $v^n \equiv x_n(a)+y_n(a)(v-a) \pmod{v^2 = 2av+1}$ where $a \geq 2$ & $((a+\sqrt{a^2-1})^n = x_n(a) + y_n(a)\sqrt{a^2-1})$*

Proof. Viewing $(v^2-2av+1)$ as a polynomial in v. it factors as: $(v-(a+\sqrt{a^2-1}))(v-(a-\sqrt{a^2-1}))$. For $v = (a+\sqrt{a^2-1})$ or $v = (a-\sqrt{a^2-1})$ the congruence is true. □

Suppose we have a Diophantaine representation of $x_n(a)$ and $y_n(a)$ say

$$F(a,x,y,n) := ((x = x_n(a) \ \& \ y = y_n(a)$$

Then we immediatey have a representatin of $u \equiv v^n \pmod{v^2-2av+1}$. It will be an absolute equality if $(2av - v^2 - 1) > u, v^n$.

Lemma 1.3.24. $x_n(b) \geq b^n \ \forall b \geq 2$

Proof. We have $(b+\sqrt{b^2-1})^n = x_n(b)+y_n(b)\sqrt{b^2-1}$. We see that $x_0(b) = 1$ and $x_1(b) = b$. Observe that $x_{n+1}(b)+y_{n+1}(b)\sqrt{b^2-1} = (x_n(b)+y_n(b)\sqrt{b^2-1})(b+\sqrt{b^2-1}) = (bx_n(b) + (b^2-1)y_n(b)) + (x_n(b) + by_n(b))\sqrt{b^2-1}$ which implies by comparison of coefficients

$$\begin{array}{rcl} x_{n+1}(b) & = & bx_n(b) + (b^2-1)y_n(b) \\ & \geq & bx_n(b) \\ \Rightarrow x_{n+1}(b) & \geq & bn+1 \end{array}$$

□

We can make $2av - v^2 - 1 > v^n$ if we froce $(2av - v^2 - 1) > x_n(v)$. So the conditions for $u = v^n$ are:

E_1**:** $F(a,x,y,n)$ represents $(a+\sqrt{(a^2-1)})^n = x + y\sqrt{a^2-1}$

E_2**:** $u \equiv x + y \pmod{v^2 = 2av+1}$ represents $u0v^n \pmod{v^2-2av+1}$

E_3**:** $(u < (2av - v^2 - 1)) \ \& \ (X < (2av - v^2 - 1))$

E_4**:** $F(v,X,Y,n)$ represents $(v+\sqrt{v^2-1})^n = (X + Y\sqrt{v^2-1}) \Rightarrow X \geq v^n$

Exercise 1.3.25. *Verify that: $(u = v^n)$ is equivalent to*

$$\exists a \exists x \exists y \exists X \exists Y (E_1 \ \& \ E_2 \ \& \ E_3 \ \& \ E_4)$$

Finally we want to represent $x_n(a)$ and $y_n(a)$ such that

$$\left(a+\sqrt{a^2+1}\right)^n = x_n(a) + y_n(a)\sqrt{a^2-1}$$

$(x_n(a), y_n(a))$ are roots of a special Fermat's equation:

$$x^2 - (a^2-1)y^2 = 1$$

because: $1 = x_n(a)^2 - (a^2-1)y_n(a)^2$ and $(a+\sqrt{a^2-1})(a-\sqrt{a^2-1}) = 1$

In the hope of representing $(x_n(a), y_n(a))$ we investigate this equation:

We can write a recurrence for $(x_{m+n}(a)$ in terms of $x_m(a)$, $y_n(a)$ and $x_n(a)$ because of $x_{m+n}(a) + y_{m+n}(a)\sqrt{a^2-1} = (x_m(a) + y_m(a)\sqrt{a^2-1})(x_n(a) + y_n(a)\sqrt{a^2-1})$

Lemma 1.3.26.

$$\begin{array}{rcl} x_{m+n} & = & x_m x_n + (a^2-1) y_m y_n \\ y_{m+n} & = & x_m y_n + x_n y_m \\ x_{n+1} & = & a x_n + (a^2-1) y_n \\ x_{n+1} & = & a y_n + x_n \end{array}$$

Lemma 1.3.27. *Say $a \geq 2$. Then*

$$x^2 - (a^2-1)y^2 = 1; x, y \geq 0 \Leftrightarrow \exists n (x = x_n(a) \ \& \ y = y_n(a))$$

Proof. "$\Leftarrow$" is clear, so suppose $x^2 - (a^2-1)y^2 = 1$ and (w.l.o.g.) $y > 1$.

$$\begin{array}{l} \Rightarrow (x - y\sqrt{a^2-1})(x + y\sqrt{a^2-1}) = 1 \\ \Rightarrow [(x - y\sqrt{a^2-1})(a + \sqrt{a^2-1})][x + y\sqrt{a^2-1})(a - \sqrt{a^2-1})] = 1 \end{array}$$

Look at the integral points give by this equation: $(ax - y(a^2-1), -x + ay)$ which is obviously satisfying the fermat equation. It is an $\mathbb{N}$-point and its "smaller" than (x, y).

- Suppose $ax < y(a^2-1)$ then $1 = x^2 - (a^2-1)y^2 < y^2 \frac{(a^2-1)^2}{a^2} - (a^2-a)y^2 = \frac{(a^2-1)y^2}{a^2}(-1) < 0$, a contradiction.
- Suppose $x < ay$ then $1 = x^2 - (a^2-)y^2 = a^2y^2 - (a^2-1)y^2 = y^2 > 1$ also a contradiction.
- Suppose $-x + ay \geq y$ then $1 = x^2 - (a^2-1)y^2 \leq (a-1)^2 y^2 - (a^2-1)y^2 = (-2a+1)y^2 < 0$, a contradiction.

□

We can see that several (say n) applications of the descent will have to stop at: $y = 1, x = a$.

$$\begin{array}{rl} & (x + y\sqrt{a^2-a})(a - \sqrt{a^2-1})^n = (a + \sqrt{a^2-1}) \\ \Rightarrow & (x + y\sqrt{a^2-a}) = (a + \sqrt{a^2-1})^{n+1} \end{array}$$

Thus, $x = x_{n+1}(a)$ & $y = y_{n+1}(a)$.

Next we show that modulo relations: $x_N(a) \equiv x_m(a) \pmod{x_n(a)}$ is almost equivalent to: $N \equiv m \pmod{n}$.

Lemma 1.3.28. *Say, $a \geq 2$ & $0 < m < n$. Then*

$$\forall N, x_N(a) \equiv x_m(a) \pmod{x_n(a)} \Leftrightarrow N \equiv \pm m \pmod{4n}$$

Proof. Let us compute $x_{i+2n} \pmod{x_n}$:

$$\begin{array}{rrcl} & x_{i+2n} + y_{i+2n}\sqrt{a^2-1} & = & (x_i + y_i\sqrt{a^2-1})(x_n + y_n\sqrt{a^2-1})^2 \\ \Rightarrow & x_{i+2n} & \equiv & x_i y_n^2 (a^2-1) \pmod{x_n} \\ \Rightarrow & x_{i+2n} & \equiv & x_i (x_n^2 - 1) \pmod{x_n} \\ \Rightarrow & x_{i+2n} & \equiv & -x_i \pmod{x_n} \end{array}$$

□

Exercise 1.3.29. *Similary, compute $x_{2n-i} \pmod{x_n}$ and proof that*

$$x_{2n-i} \equiv -x_i \pmod{x_n}$$

Therefor, we have:

- $\{x_0 x_1, \ldots, x_{n-1}\}$
- $\{x_n, \ldots, x_{2n-1}\} \equiv_{x_n} \{-x_n, -x_{n-1}, \ldots, x_1\}$
- $\{x_{2n}, \ldots, x_{3n-1}\} \equiv_{x_n} \{-x_0, -x_1, \ldots, -x_{n-1}\}$
- $\{x_{3n}, \ldots x_{4n-1}\} \equiv_{x_n} \{x_n, x_{n-1}, \ldots, x_1\}$

Exercise 1.3.30. *Show that $x_0 < x_1 < \cdots < x_{n-1} < x_n$ using 1.3.26*

So x_m appears at $\pm m \pmod{4n}$ places. Now do the same thing for y:

Lemma 1.3.31. *Say, $a \geq 2$ & $n \geq 1$ then $\forall N : y_N(a) \equiv 0 \pmod{y_n(a)} \Leftrightarrow N \equiv 0 \pmod{n}$*

Exercise 1.3.32. *proof this lemma.*

We can also look at $y_N \pmod{y_n^2}$:

Lemma 1.3.33. *Say, $a \geq 2$ & $n \geq 1$, then $\forall N, y_N(a) \equiv 0 \pmod{y_n(a)^2} \Leftrightarrow N \equiv 0 \pmod{n y_n}$.*

Proof. "$\Rightarrow$" Suppose $y_n^2 | y_N$ then $n|N$ and so $\exists k, N = kn$:

$$\begin{array}{rcl} \Rightarrow \quad (x_N + y_N\sqrt{a^2-1}) & = & (x_{kn} + y_{kn}\sqrt{a^2-1}) = (x_n + y_n\sqrt{a^2-1})^k \\ & = & \sum_{i=0}^{k} \binom{k}{i} x_n^{k-1} \cdot (y_n\sqrt{a^2-1})^i \\ \Rightarrow \quad y_n & \equiv & k x_n^{k-i} y_n \pmod{y_n^2} \end{array}$$

Since $y_n^2 | y_N$ we get $y_n | k x_n^{k-1} \Rightarrow y_n | k \Rightarrow n y_n | N$. "$\Leftarrow$" Suppose $n y_n | N$. Then $\exists k : N = kn$ & $y_n | k$ and $y_N \equiv 0 \pmod{y_n^2}$ by 1.3. □

Now look $\pmod{a-1}$:

Lemma 1.3.34. *Say $a \geq 2$ & $n \geq 1$ then $(x_n, y_n) \equiv (1, n) \pmod{a-1}$.*

Proof.
By 1.3.26:

$$\begin{array}{rcl} x_{i+1} & = & a x_i + (a^2-1) y_i \\ x_{i+1} & = & x_i + a y_i \end{array}$$

$$\begin{array}{rclcl} x_{i+1} & \equiv & x_i \pmod{a-1} & \Rightarrow & x_i \equiv 1 \pmod{a-1} \\ x_{i+1} & \equiv & x_i + y_i \pmod{a-1} & \Rightarrow & y_i \equiv i \pmod{a-1} \end{array}$$

Simply 1.3.26:

Lemma 1.3.35.

$$a \equiv a' \pmod{b} \Rightarrow (x_n(a), y_n(a)) \equiv (x_n(a'), y_n(a')) \pmod{b}$$

Exercise 1.3.36. *proof this lemma*

Now we are ready to give a Diophantine representation of $F(a,x,y,n) := (a + \sqrt{a2-1})^n = x + y\sqrt{a^2-1}$. We certainly need $F_1 : x^2 - (a^2-1)y^2 = 1$. But how do we ensure that (x,y) is the n-th point in F_1? So we can put $y \equiv n \pmod{a-1}$, but it is not enough as n can be much larger hen a. The idea is to construct sort of an n-th (X,Y) point on another Fermat-equation $F_2 : X^2 - (A^2-1)Y^2 = 1$ which has a large A. Then connect the points (x,y) and (X,Y) by a third point (U,V) satisfying $F_3 : U^2 - (a^2-1)V^2 = 1$ such that $F(a,x,y,n)$ is forced.

Say $(x,y) = (x_m(a), y_m(a)); (U,V) = (x_N(a), x_N(a)), (X,Y) = (x_M(A), y_M(A))$.

Force M to be sort of n $F_4 : Y \equiv n \pmod{A-1}$ which implies 1.3.35 $\Rightarrow M \equiv n \pmod{A-1}$

connect (X, Y) to (x, y) via (U, V)

$$F_5 : 0 < x < U \ \& \ X \equiv x \pmod{U} \ \& \ A \equiv a \pmod{U}$$

and with 1.3.27 it follows that $x_M(a) \equiv x_M(a) \equiv x_m(a) \pmod{y_N(a)}$ and by 1.3.31 it follows that $M \equiv \pm m \pmod{4N}$.

connect these two congruences $F_6 : V \equiv 0(mdy^2)$ by 1.3.34 it follows that $my|N$. $F_7 : A - 1 \equiv 0 \pmod{4y}$ which implies $4y|(A-1), 4N \Rightarrow n \equiv \pm m \pmod{4y}$

make the congruence absolute $F_8 : n \leq y$ which implies $m \leq y \Rightarrow n = m$

Exercise 1.3.37. *Verify that* $(a \geq 2)$ & $(a + \sqrt{a^2-1})^n = (x + y\sqrt{a^2-1})$ *is equivalent to* $(a \geq 2)$ & $\exists A \exists X \exists Y \exists U \exists V (F_1 \ \& \ \ldots \ \& \ F_8)$

□

Chapter 2

P, NP and beyond

We now consider problems that have obious algorithms that run in finite time and will study the resources required by an algorithm. Examples for resources are:

time which is the number of transitions of a turing machine M till it stops on a given input x. We say that is the "time taken by M on x".

space which is the number of cells used on the tape

randomness which is a more exotic resources, describing the number of random bits an algorithm uses

Time taken by M is viewed as a function $\text{time}_M : \mathbb{N} \to \mathbb{N}, n \mapsto \text{time}_M(n)$ where $\text{time}_M(n) := \max_{|x|=n}\{\text{time taken by } M \text{ on } x\}$. As time_M is dependent on low-level details of M, it is quite messy to specify time_M exactly. So we just applroximate $\text{time}_M(n)$ by asymptotic. Two functions $f, g : \mathbb{N} \to \mathbb{N}$ can be in the following relations:

a) $f = O(g) \Leftrightarrow \exists c, N > 0 \forall n \geq N. f(n) \leq cg(n)$

b) $f = \Omega(g) \Leftrightarrow g = O(f)$

c) $f = \Theta(g) \Leftrightarrow f = O(g)$ & $g = O(f)$

d) $f = o(g) \Leftrightarrow \forall \epsilon \in R^{>0} \exists N > 0 \forall n \geq N$

e) $f = \omega(g) \Leftrightarrow g = o(f)$

Example 2.0.38. $10n^2 + 100\log(n) = O(n^2), \Theta(n^2), \omega(n^{1.99}) \neq \omega(n^2)$

Definition 2.0.39. *We say that*

- *g is an upper bound on f if $f = O(g)$*
- *f is a lower bound on g if $g = \Omega(f)$*

Exercise 2.0.40. *Show that the high-school method to add two integers takes time $\Theta(n)$.*

2.1 Polynomialtime

For a function $T : \mathbb{N} \to \mathbb{N}$ we define a set of languages (which is equivalent to decision problems) that have algorithms of time $O(T(n))$:

$$\begin{aligned}\text{Dtime}(T) := \{L \subset \{0,1\}^* \mid\ & L \text{ is decided by a turing machine } M \\ & \& \text{ time}_M(n) = O(T(n))\}\end{aligned}$$

Which algorithms should we call "efficient" now? Algorithms that run in time $O(|x|^n)$ for a constant n are called efficient. The set of all these problems is called P:

$$P := \bigcup_{c \in \mathbb{N}} \text{Dtime}(n^c)$$

It stands for polynomial-time algorithms. We call an algorithm or a turing machine M deterministic polynomial-time if it is solving a problem $L \in P$. There are some tricky issues with polynomial time: For $\text{time}_{M_1} = f_1(n) = n^{100}, \text{time}_{M_2}\, f_2(n) = n^l glglgn$ the turing machine M_1 is polynomial-time while M_2 is not. M_2 is more than polynomial-time. But

$$\text{time}_{M_2}(n) > \text{time}_{M_1}(n) \Leftrightarrow n > 2^{2^{2^{100}}}$$

Nevertheless P is the most eleganz way to study inherent complexity of natural problems.

2.2 NP: Nondeterministic polynomial-time

Let us start with some example problems:

Travelling Salesman Problem (TSP) Suppose we are given a complete graph on n nodes $\{1,\ldots,n\}$ with distances $\{d_{ij} \in \mathbb{N} | 1 \le i,j \le n\}$. Is there a closed circuit visiting each node exactly once of length $\le k$?

Subset Sum Given a set $S = \{s_1,\ldots,s_m\} \subseteq \mathbb{N}$ and $t \in \mathbb{N}$. Is there a subset $T \subseteq S$ that sums to t?

Integer Programming Given m linear inequalities, over $\mathbb{Z}$ in $x_1,\ldots,x_n$:

$$a_{i1}x_1 + \ldots + a_{im}x_m \le b_i \quad \forall 1 \le i \le m$$

Is there a common point $(x_1,\ldots,x_m) \in \mathbb{Z}^m$?

Exercise 2.2.1. *Proof that*

a) "TSP" can be solved in $O(n!) \approx O(2^{nlog(n)})$ *time*

b) "Subset Sum" can be solved in $O(2^m)$ *time*

c) "Integer Programming" can be solved in $O(2^{m^2})$ *time*

All these algorithms are in

$$EXP := \bigcup_{c \in \mathbb{N}} \text{Dtime}(2^{n^c})$$

Anything better for these three problems is still unknown!

Exercise 2.2.2. *Show that a given candidate solution for "TSP", "Subset Sum" or "Integer Programming" can be verified in P.*

This was observed and the class NP was defined by Cook (1971) and Levin (1973).

Definition 2.2.3. *A language $L \subset \{0,1\}^*$ is in NP if $\exists c \in N$ and a polynomial-time turing-machine M s.t. $\forall x \in \{0,1\}^*$:*

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{|x|^c}, M(x,u) = 1$$

This u satisfying $M(x,u) = 1$ is called a certficate for x with respect to M and M is called a verifier of L.

Lemma 2.2.4. *"Subset Sum" is in NP*

Proof.

language

$$L := \left\{(S,t) \mid S \subseteq \mathbb{N}, t \in \mathbb{N}, \exists T \subset S, \sum_{x \in T} x = t\right\}$$

input $x := (S,T)$

certificate $u := T$ for which $\sum_{x \in T} x = t$ & $T \subseteq S$

verifier $M :=$ the turing machine that outputs 1 on input (S,t,T) iff

$$T \subseteq S \ \& \ \sum_{x \in T} x = t$$

constant $c := 1$

□

Exercise 2.2.5. *Proof that $P \subseteq NP$ (note that u can be just taken empty)*

Lemma 2.2.6. $NP \subseteq EXP$

Proof. Let $L \in NP$, let M be its verifier (of time n^d) and c s.t. $x \in L$ iff $\exists u \in \{0,1\}^{|x|^c}, M(x,u) = 1$. Define M' to be the turing machine that scans an u of length $|x|^c$, simulates M on (x,u) and outputs 1 iff $M(x,u) = 1$ for some u. Note that M' runs in time $2^{|x|^c}|x|^d < 2^{|x|^{c+1}}$ and that M' accpets x iff $x \in L$. So $L \in \text{Dtime}(2^{|x|^{c+1}})$. □

Remark 2.2.7. *Its not known whether $P = NP$ or $NP = EXP$.*

Definition 2.2.8. *A Nondeterministic Turing Machine (NDTM) N is a Turing Machine (1.2.1) with two transition functions, so $N = (\Gamma, Q, \delta_0, \delta_1)$ (+ infinite tapes). At any configuration $C = [s,p,b]$ it's transition is no more unique and it can follow δ_0 or δ_1 each step. N is said to accept $x \in \{0,1\}^*$ iff there is a non-empty set of options for steps leading to the halt and the accept state. The time taken by N on x is $\min_{paths}\{\#$ steps taken to halt on $x\}$.*

We cannot really identify this with a physical device (unlike we did with turing-machine , which are actually computers), so a non-deterministic turing-machine is really an abstract machine. non-deterministic turing-machines motivate a class, like Dtime:

Definition 2.2.9. *Let T be a functions $\mathbb{N} \to \mathbb{N}$, define*

$$\begin{aligned}\text{Ntime}(T(N)) \quad := \quad & \{L \subseteq \{0,1\}^* \mid L \textit{ is decided by a non-deterministic turing-machine } N \\ & \& \; time_N(n) = O(T(n))\}\end{aligned}$$

Theorem 2.2.10.

$$NP = \bigcup_{c \in \mathbb{N}} Ntime(n^c)$$

Proof. First we show that $NP \subseteq \bigcup_{c\in\mathbb{N}} Ntime(n^c)$: Let $L \in NP$. Then L has a deterministic polynomial-time verifier M, s.t. $x \in L$ iff $\exists u \in \{0,1\}^{|x|^c}, M(x,u) = 1$. Define a non-deterministic turing-machine N that has transition functions δ_0 and δ_1 s.t. on input x, N writes at each transition either 0 (by δ_0) or 1 (by δ_1). N does this for $|x|^c$ many steps, call this output w. Then N should simulate $M(x,w)$ and copies its output. So $L \in Ntime(nc + 1 + time_M(x))$.

Next we show $\bigcup_{c\in\mathbb{N}} Ntime(n^c) \subseteq NP$. Let $L \in Ntime(n^c)$. Then there exists a non-deterministic turing-machine N that decides L in time $O(n^c)$. $x \in L$ iff $\exists$ path p in N (which can be views as string of length $O(|x|^c)$) s.t. N accepts x in p which has a deterministic polynomial-time turing-machine given (x,p) as input, simulate N following p. So $L \in NP$. □

Definition 2.2.11. *A boolean formular in cinjunctive normal form (CNF) is*

$$\phi(x_1, \dots, x_n) = \bigwedge_i \left(\bigvee_j v_{ij} \right)$$

where the literals $v_{ij} \in \{x_1, \dots, x_n, \overline{x_1}, \dots \overline{x_n}\}$. The Terms in the brakets are called clause. A boolean formula ϕ is called satisfiable if there exists a valuation v of the variables s.t. $\phi(v) = 1$.

NP has a "hardest" problem (with respect to efficiency, not to computability).

Definition 2.2.12. *SAT: Given a boolean formula ϕ in CNF, decide if ϕ is satisfiable. The corresponding language is:*

$$SAT := \{\phi \mid \phi \textit{ is in CNF and is satisfiable}\}$$

Example 2.2.13.

$$(x_1 \vee \overline{x_2} \vee x_3) \; \& \; (\overline{x_2} \vee x_3)$$

(so it's a combination of $\vee$, & , $\overline{\cdots}$)

Lemma 2.2.14.

$$SAT \in NP$$

Proof. Given a boolean formula ϕ in CNF. The valuation v is the certificate and the verifier is: evaluate ϕ at v. □

Lemma 2.2.15. *$\forall L \in NP$, L can be "efficiently reduced" to SAT.*

Proof. (The proof is really what we did in the beginning of proof of Hilbert's 10th problem) As $L \in NP$ there exists a polynomial-time verifier M s.t.

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{|x|^c}, M(x,u) = 1$$

Idea: We want to capture the computation of M on (x,u) step-by-step in a CNF formula ϕ_x s.t. ϕ_x is satisfiable iff $\exists u$ with $M(x,u) = 1$ in other words:

$$\phi_x \in SAT \Leftrightarrow x \in L$$

With each configuration C of M we associated an array of variables:

$$[s(C), p(C), a_0(C), \dots, a_{T-1}(C)]$$

but now we just know T as a functions of $|x|$. Final formula looks like:

$$\phi_x(\dots) \coloneqq Start(C_1, x) \mathbin{\&} Compute(C_1, C_2) \mathbin{\&} Stop(C_2)$$

where $Compute(C_1, C_2)$ asserts that there is a sequence of configurations $C_1 = g_0, \dots, g_{T-1} = C_2$ s.t. each step is valid (follows δ_M):

$$(g_0 = C_1) \mathbin{\&} (g_{T-1} = C_2) \mathbin{\&} (\forall i < T)(\bigvee_{I \in \delta_M} Step_I(g_i, g_{i+1}))$$

and $Step_I(g_i, g_{i+1})$ asserts that the transition from g_i to g_{i+1} follows the instructions I: $(q,b) \mapsto (q', b', \epsilon)$. For $\epsilon =$ "Stay":

$$s(g_i) = q \mathbin{\&} s(g_{i+1}) = q'$$

$$\mathbin{\&} \exists k(p(g_i) = p(g_{i+1}) = k \mathbin{\&} a_k(g_i) = b \mathbin{\&} a_k(g_{i+1}) = b' \mathbin{\&}$$
$$(\forall j < T)(j = k \vee a_j(g_i) = a_j(g_{i+1})))$$

Exercise 2.2.16. *a) Observe that "$a = b$" can be written as & s of $a_i = b_i$ where are a_i, b_i are bits.*

$$a_i = b_i \Leftrightarrow (a_i \vee \overline{b_i}) \mathbin{\&} (\overline{a_i} \vee b_i)$$

b) $(\forall i < T) E_i \equiv E_0 \mathbin{\&} \dots \mathbin{\&} E_{T-1}$

c) We can convert M to an oblivious turing-machine M' (i.e. on an input x the head-position at the i-th step of M' is just a function of $(|x|, i)$) and then construct ϕ_x.

d) $x \in L$ iff ϕ_x is satisfiable.

□

Theorem 2.2.17. *SAT is NP-hard.*

Remark 2.2.18. *SAT can be "efficiently reduced" to $3SAT$ (where $3SAT \coloneqq \{\phi \mid \phi$ is a 3 CNF and is satisfiable$\}$. We say that ϕ is in 3 CNF if $\phi = \bigwedge_i (v_{i1} \vee v_{i2} \vee v_{i3})$).*

Proof. Idea: A clause like $x_1 \vee \overline{x_2} \vee x_3 \vee \overline{x_4}$ is satisfiable iff $(x_1 \vee \overline{x_2} \vee z)$ & $(\overline{z} \vee x_3 \vee \overline{x_4})$ is satisfiable. So ϕ can be converted into an equivalent 3 CNF wth more variables and more clauses (but not much more, it's only a polynomial blow-up). □

$3SAT$ is a hardest problem in NP. We now formalize this feature:

Definition 2.2.19. *We say that a problem A reduces to a problem B via a complexity class C (in symbols $A \leq_C B$) iff there is a turing machine M in C s.t. $x \in A$ iff $M(x) \in B$.*

Definition 2.2.20. *A problem A is Karp reducible to B iff $A \in_P B$ (where P is the polynomial-time complexity class)*

Definition 2.2.21. *We say that a problem B is NP-hard iff $\forall A \in NP : A \leq_P B$ and we say that it is NP-complete iff B is NP-hard and $B \in NP$.*

Exercise 2.2.22.

- *$A \leq_P B$ & $B \leq_P C \Rightarrow A \leq_P C$*
- *If A is NP-hard & $A \in P \Rightarrow P = NP$*
- *If A is NP-hard & $A \leq_P B \Rightarrow B$ is NP-hard*
- *If A and B are NP-complete $\Rightarrow A \leq_P B$ & $B \leq_P A$*

Corrolar 2.2.23. *Thus NP-complete problems are the hardest in NP.*

Lemma 2.2.24.

$$3SAT \leq INTEGER - PROGRAMMING$$

Proof. Let $\phi = C_1$ & $\dots$ & C_m and $x_{i,j} \in \{x_j, \overline{x_j}\}$ the variables in C_i for $i \in \{1, \dots, m\}$, $j \in \{1,2,3\}$ and define for $j \in \{1,2,3\}$

$$\begin{array}{lclclcll} \text{if } x_{i,j} & = & x_j & \text{define} & z_{i,j} & = & y_{i,j} & \in \mathbb{Z} \\ \text{if } x_{i,j} & = & \overline{x_j} & \text{define} & z_{i,j} & = & 1 - y_{i,j} & \in \mathbb{Z} \end{array}$$

Now for every clause C_i we generate the following inequalities:

$$\begin{array}{rcl} z_{i,1} + z_{i,2} + z_{i,3} & \geq & 1 \\ 1 \geq y_{i,1} & \geq & 0 \\ 1 \geq y_{i,2} & \geq & 0 \\ 1 \geq y_{i,3} & \geq & 0 \end{array}$$

Example: We transform $(x_1 \vee \overline{x_2} \vee \overline{x_3})$ (where $x_1, x_2, x_3 \in \{T, F\}$) into

$$\begin{array}{rcl} y_1 + (1 - y_2) + (1 - y_3) & \geq & 1 \\ 1 \geq y_1 & \geq & 0 \\ 1 \geq y_2 & \geq & 0 \\ 1 \geq y_3 & \geq & 0 \\ y_1, y_2, y_3 \in \mathbb{Z} & & \end{array}$$

Exercise 2.2.25. *Show that the integer-program, defined above, is solveable iff ϕ is satisfiable.*

□

Definition 2.2.26. *Quadratic equality over finite fields:*

$QUADEQN := \{S \mid S$ *is a set of quadratic equations over a finite field* & S *has a root*$\}$

Lemma 2.2.27.

$$3SAT \leq_P QUADEQN \;\&\; QUADEQN \in NP$$

Proof. Let $\phi = C_1 \;\&\; \dots \;\&\; C_m$ and $x_{i,j} \in \{x_j, \overline{x_j}\}$ the variables in C_i for $i \in \{1, \dots, m\}, j \in \{1,2,3\}$ and define for $j \in \{1,2,3\}$

Example: We transform $(x_1 \vee \overline{x_2} \vee \overline{x_3})$ (where $x_1, x_2, x_3 \in \{T, F\}$) into $1 - (1 - x_1)x_2x_3$ with $x_1, x_2, x_3 \in \mathbb{F}_2$.

Define $\psi(\phi) := \prod_{i=1}^{m} \phi(C_i)$, now we see that $\psi(\phi) = 1$ has an $\mathbb{F}_2$-root iff ϕ is satisfiable.

How to reduce the degree (from $3m$ to 2)?

$$1 - (1 - x_1)x_2x_3 \mapsto \begin{cases} 1 - (1 - x_1)z_1 & = & 1 \\ z_1 & = & x_2x_3 \end{cases}$$

□

2.3 co-Classes

Definition 2.3.1. *To a language* $L \subseteq \{0,1\}^*$ *we define the co-language* $\overline{L}$ *by*

$$\overline{L} := \{0,1\}^* L$$

Definition 2.3.2.

$$coNP := \{L \subseteq \{0,1\}^* \mid \overline{L} \in NP\}$$

Remark 2.3.3. *If* $L \in NP$ *then given an* x*, it is "easy" to verify whether* $x \notin L$ *(or equivalently* $x \in \overline{L}$*).*

Definition 2.3.4. *A DNF formular is a formular in the form:*

$$\bigvee_i \left(\bigwedge_j^{n_i} x_{ij} \;\&\; \bigwedge_j^{m_i} \overline{x_{ij}} \right)$$

Definition 2.3.5. *Problem: Tautology*

$TAUT := \{\phi \mid \phi$ *is a DNF formular and a tautology (i.e. true for all evaluations)*$\}$

Remark 2.3.6.

$$TAUT \in coNP$$

Proof. Given $\phi(x_1, \dots, x_n)$, $\phi \notin TAUT$ iff $\exists v : \phi(v) = False$. □

Theorem 2.3.7. $TAUT$ *is coNP-hard.*

Proof. Given $L \in coNP$ we can want to reduce L to $TAUT$: We knot that there exists a polynomial-time turing-machine M s.t. $x \in \overline{L}$ iff $\exists u : M(x, u) = 1$. Using the idea in 2.2.17 we construct a boolean formular ϕ_x s.t. $x \in \overline{L}$ iff ϕ_x is satisfiable. So $x \in L$ iff ϕ_x is unsatsisfiable. And $\neg\phi_x$ is a tautology. In symbols:

$$L \leq_P TAUT$$

□

Definition 2.3.8.

$$NP \cap coNP := \{L \subseteq \{0,1\}^* \mid L \in NP \,\&\, L \in coNP\}$$

Remark 2.3.9. *$P \subseteq NP \cap coNP$ but it is not known whether if this is an equation or a strict subset-relation.*

Exercise 2.3.10. *"The decision version of integer factoring" $\in NP \cap coNP$*

Remark 2.3.11. *It is unknown if*

$$GraphIso \in NP \cap coNP$$

Theorem 2.3.12.

$$NP \neq coNP \Rightarrow P \neq NP$$

Proof. If $NP \neq coNP$ but $P = NP$ than $P \neq coP = P$ which is a contradiction. □

Remark 2.3.13. *But it's an open question whether $NP = coNP$ or whether $NP = coNP \Rightarrow P = NP$.*

Definition 2.3.14.

$$1^n := \underbrace{1\dots1}_{n-times}$$

Remark 2.3.15. *Gödels's question:*

$$Theorems \;:=\; \{(\phi, 1^n) \mid \phi \text{ is a mathematical statement for which there is a proof of length } \leq n\}$$

First notice that $Theorems \in NP$ and that it is NP-hard. If $P = NP$ then $Theorems \in P$ and mathematicians are obsolete.

2.4 EXP and $NEXP$

Definition 2.4.1.

$$EXP := \bigcup_{C \in \mathbb{N}} \mathrm{Dtime}(2^{n^C})$$

is the class of exponential-time problems, and

$$NEXP := \bigcup_{C \in \mathbb{N}} Ntime(2^{n^C})$$

is the class of nondeterministic exponential-time problems.

Remark 2.4.2. *$P \subseteq NP \subseteq EXP \subseteq NEXP$ and it is unknown whether any of this relations are strict. It is known that $P \neq EXP$ and $NP \neq NEXP$.*

Theorem 2.4.3.

$$P = NP \Rightarrow EXP = NEXP$$

Proof. Let $P = NP$ and $L \in NEXP$, say $L \in Ntime(2^{n^C})$ for some $n, C \in \mathbb{N}$ and M is a polynomial-time turing-machine s.t.

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{2^{|x|^C}} : M(x,u) = 1$$

Define a language $L' := \{(x, 1^{|x|^C}) \mid x \in L\}$ (this trick is called padding technice). And we see that $L' \in NP$ with the same verifier as M and the same certificate as in L. So $L' \in P \Rightarrow L \in EXP \Rightarrow NEXT \subseteq EXP \Rightarrow NEXP = EXP$. □

Remark 2.4.4. *It's unknown whether $EXP = NEXP \Rightarrow P = NP$.*

Definition 2.4.5. *Let $f : \mathbb{N} \to \mathbb{N}$, define*

$$\text{Space}(f(n)) := \{L \subseteq \{0,1\}^* \mid L \textit{ is decided by a turing machine that uses } O(f(n)) \textit{ cells/space}\}$$

and then we can define

$$P_{Space} := \bigcup_{c \in \mathbb{N}} \text{Space}(n^c)$$

2.5 Hierarchy Theorems

Now we will show some "easy" complexity class sparations: given "strictly more" resources (time, space, nondeterminism) turing machines can soolve more problems.

Theorem 2.5.1. *Let $f, g : \mathbb{N} \to \mathbb{N}$ s.t. $g(n) = \omega(f(n)^2)$ then* $\text{Dtime}(f(n)) \subsetneq \text{Dtime}(g(n))$.

Proof. Idea is the use of "diagonalization", i.e. simulating M_y on y. Consider the turing machine D which does the following on an input $x \in \{0,1\}^*$:

- Output 0 if x is not a description of a turing machine
- else simulate the encoded turing machine M_x with x as input. Simulate $g(|x|)$ steps:
  - if it does not get an output in $\{0,1\}$ then output 0
  - else if it does get an output $M_x(x)$ output $1 - M_x(x)$

Notice that D desides a language, say L, in time $g(n)$. So $L \in \text{Dtime}(g(n))$. Suppose $L \in Dtime(f(n))$ and let M be a turing machine deciding L in time $\leq c \cdot f(n) \forall n \geq n_0$ for a constant n_0.

Remark: There exist infinitely many strings $y \in \{0,1\}^*$ that describe M (add garbage steps and variables to M and encode it againt). Pick a "large" $y \in \{0,1\}^*$ describing M with $g(|y|) \geq (c \cdot f(|y|))^2$ and $y \geq n_0$. What is $D(y)$? $M_y = M$ on y gives an answer in $\{0,1\}$ in $\leq c \cdot f(|y|)$.

Exercise 2.5.2. *Show that D can simulate "s steps of M" in $\leq s^2$ steps*

So D can be simulate $c \cdot f(|y|)$ steps of $M_y(y)$ in $g(|y|)$ steps so $D(y) = 1 - M_y(y) = 1 - M(y)$. This contradicts. D and M deciding the same language L. So $L \notin \text{Dtime}(f(n))$. □

Theorem 2.5.3. *Let* $f, g : \mathbb{N} \to \mathbb{N}$ *s.t.* $g(n) = \omega(f(n))$ *then* $\text{Space}(f(n)) \subsetneq \text{Space}(g(n))$.

Exercise 2.5.4. *Proof the preceding theorem.*

Theorem 2.5.5. *Let* $f, g : \mathbb{N} \to \mathbb{N}$ *s.t.* $g(n) = \omega(f(n))$. *Then* $\text{Ntime}(f(n)) \subsetneq \text{Ntime}(g(n))$.

Proof. The idea is to use "lazy diagonalisation": D will be "mostly" be identical to a non-deterministik turing machine M_y and differ only for "very large" y. Define a turing machine D which acts on input $x \in \{0,1\}^*$:

- If $x \notin \{1\}^*$ then output 0
- else
  - if $s(i) < n < s(i+1)$ then simulate the non-deterministik turing machines $M_i(1n+1)$ for $g(|x|)$ steps
  - if $n = s(i+1)$ then output 1 if $M_i(1^{1+s(i)})$ recets in $g(i + s(i))$ steps and output 0 if it accepts.

If we choose $2^{g(1+s(i))} < g(s(i+1))$ then D decides a language $L \in Ntime(g(n))$.

Remark: for $g(n) = n$ the function is $21 + s(i) < s(i+1)$ which means that s is very rapidly growing. Again suppose that $L \in \text{Ntime}(f(n))$. First suppose it is, let M be a non-deterministic turing machine deciding L in time $\leq c \cdot f(n)$, $\forall n \geq n_0$. We remark again there are infinitely many strings $y \in \{0,1\}^*$ describing M. Pick a "large" y desciribing for M. Notice that $M = M_y$ and for all $n \in (s(y), s(y+1)) \cap \mathbb{Z}$ we see $d(1^n) = M_y(1^n) = M(1^n)$. By step (2) for all $n \in (s(y), s(y+1)) \cap \mathbb{Z}$, $D(1^n) = M_y(1^{n+1}) = M(1^{n+1})$.

$\Rightarrow M(1^{1+s(y)}) = D(1^{s(y)+1}) = D(1^{s(y)+2}) = \ldots = D(1^{s(y+1)})$

But by step (3): $D(1^{s(y+1)}) \neq M_y(1^{1+s(y)})$ which is a contradiction and

$$L \notin \text{Ntime}(f(n))$$

□

These Hirarchy theorems are based on:

a) corresponding turing machines can be enumerated

b) they can be simulated by a turing machine

Definition 2.5.6.

$$PSPACE := \bigcup_{c \in \mathbb{N}} \text{Space}(n^c)$$

$$EXPSPACE := \bigcup_{c \in \mathbb{N}} \text{Space}(2^{n^c})$$

Exercise 2.5.7.

$$P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq NEXP \subseteq EXPSPACE$$

Corrolar 2.5.8. *(from the Hirarchy theorems)*

- $P \neq EXP$

- $NP \neq NEXP$
- $PSPACE \neq EXPSPACE$

Are there "expected" to be only NP-hard problems in NP / P?

Theorem 2.5.9. *(Ladner's Theorem) If $P \neq NP$ then $\exists L \in NP - P$ s.t. L is not NP-complete.*

Proof. $P \neq NP \Rightarrow SAT \notin P$ and we "pad" SAT to get the promised L:

$$SAT_H := \left\{\phi 01^{|\phi|^{H(|\phi|)}} \mid \phi \text{ is a satisfiable formular in CNF}\right\}$$

for any function $H : \mathbb{N} \to \mathbb{N}$. First we shot that SAT is NP-complete for

$$\lim_{n\to\infty} H(n) = \infty$$

Suppose that H is unbounded but $SAT \leq_P SAT_H$. This means that for any CNF ψ there is a $\phi(\psi)$ by the polynomial reduction with $\psi \mapsto \phi 01|\phi|H(|\phi|)$. If $|\psi| = n$ then

$$\begin{array}{rrcl} & |\phi| + |\phi|^{H(|\phi|)} & \leq & n^C \\ \Rightarrow & |\phi| & \leq & n^{\frac{c}{H(|\phi|)}} \\ \Rightarrow & |\phi| & = & o(n) = o(|\psi|) \end{array}$$

Repeating this reduction $|\psi|$ times we get a constant sized formula whose satisfiability can be tested trivially. so $SAT \in P$, a contradiction. So for $H(n) \to \infty$ SAT_H is not NP-complete.

Secondly define H s.t $H(n) \to \infty$ and show that $SAT_H \neq P$: Let $H(n)$ be the smallest $i < \log\log n$ s.t. $\forall x \in \{0,1\}^{\leq \log n}$ and turing machine M_i generated by i accept x in $i|x|^i$ steps iff $x \in SAT_H$. If there is no such i then $H(n) := \log\log n$. This is an iteratie definition of H (it's not circular). Remark following exercise (not needed for the proof):

Exercise 2.5.10. *Such an $H(n)$ is computable in $o(n^3)$.*

Now suppose that $SAT_H \in P$ and M is a turing machine solving it in time $\leq cn^c$ for some constant c. Now let j be s.t.

- M is generated by j: $M = M_j$
- $j > c$

Thus M_j decides SAT_H in n^j steps implying by the definition of H that: $H(n) \leq j \quad \forall n$ with $\log\log j$.

$\Rightarrow$ SAT_H is equivalent to SAT (up to polynomial time computations)
$\Rightarrow$ $SAT \in P$ a contradiction
$\Rightarrow$ $SAT_H \notin P$

At last we have to show that $\lim_{n\to\infty} H(n) = \infty$. Since $SAT_H \notin P$, for any j describing a turing machine M_j there is an $x_j \in \{0,1\}^*$ s.t. M_j cannot decide $x_j \in SAT_H$ in time $j|x|^j$. Then the definition of H implies that $\{n \mid H(n) = j_0\}$ is finite, in fact $\forall n : n <\leq 2^{|}x_j|$. So $H(n) \to \infty$ for $n \to \infty$ and finally $SAT_H \notin P$, SAT_H is not NP-complete and $SAT_H \in NP$. □

Can one proof that $P \neq NP$ by these diagonalization techniques?

Definition 2.5.11. *We call a (non-deterministic) turing machine M an oracle (non-deterministic) turing machine using a language O if M has three special states q_{query}, q_{yes} and q_{no} and a special oracle tape s.t. when M enters the state q_{query} with string x on the oracle tape, in one step it moves to the state q_{yes} (or q_{no}) if $x \in 0$ (or $x \notin 0$).*

Definition 2.5.12.

$$\begin{array}{rcll} P^O & := & \{L & | \quad L \text{ has a polynomial-time oracle turing-machine using } O\} \\ NP^O & := & \{L & | \quad L \text{ has a polynomial-time oracle non-deterministic} \\ & & & \quad \text{turing machine using } O\} \end{array}$$

Exercise 2.5.13. *a)* $\overline{SAT}, TAUT \in P^{SAT}$

b) If $O \in P$ then $P^O = P$

c) With

$$ExpCom := \{(M, x, 1^n) \mid \textit{the turing machine } M \textit{ accepts } x \textit{ in } \leq 2^n \textit{ steps}\}$$

then

$$EXP = P^{ExpCom} = NP^{ExpCom}$$

2.6 can $P \neq NP$ be shown by diagonalization techniques?

The seperation proofs we saw till now also hold under any fixed oracle O, e.g.

$$\text{Dtime}(f(n))^O \neq \text{Dtime}(f(n)^3)^O$$

Definition 2.6.1. *Proofs about turing machines that also hold for every oracle O are called relativizing proofs.*

We will now show, that diagonalization proofs are not able to show $P \neq NP$ or in other words: A proof of $P \neq NP$ will be non-relativizing.

Theorem 2.6.2. *(Baker-Gill-Solovay) $\exists$ oracles A and B s.t. $P^A = NP^A$ and $P^B \neq NP^B$.*

Proof. Define

$$A := \{(M, x, 1^n) \mid \text{a turing machine } M \text{ accepts } x \text{ in less than } 2^n \text{ steps}\}$$

Exercise 2.6.3. *Proof that $EXP = P^A = NP^A$.*

We will construct B by diagonlization. For any language B let

$$U_B := \{1^m \mid \exists x \in B \text{ of length } m\}$$

and we will construct B s.t. $U_B \in NP^B$ but $U_B \notin P^B$. First note that there is an enumeration of oracle turing machines:

$$\{M_i \mid i \text{ describes the oracle turing machine } M_i\}$$

ordered accordiing to the ncreasing order of i. we will define B iteratively. Initially $B = \emptyset$ and in the i-th iteration we already have defined a finite number of strings in or out of B and let n_i be an integer greater than the length of any string whose status is already defined. Run M_i^B on 1^{n_i} for a maximum of $2^{n_i} - 1$ steps:

(1) if M_i queries B on a string whose status is undetermined, we define it "out of B".

(2) if M_i queries B on a string whose status is known, be consistent

(3) eventually in $2^{n_i} - 1$ steps if M_i^B accepts 1^{n_i} then we define all strings of length n_i "out of B".

(4) if M_i^B rejects 1^{n_i} then we put an undetermined string of length n_i (a string which status is unknown right now) in B (that is possible because M_i queries $2^{n_i} - 1$ strings at maximum)

Exercise 2.6.4. *Show that there is a string $x \in \{0,1\}^{n_i}$ at the end of the i-th iteration for which M_i^B does not decide $x \in U_B$ in $2^{n_i} - 1$.*

Thus if $U_B \in P^B$ then pick a large enough j s.t. M_j^B decides U_B in polynomial time. This contradicts the definition of B. So $U_B \notin P^B$ while $U_B \in NP^B$ which implies that $P^B \neq NP^B$. □

2.7 More on space complexity

Recall that $NP \subseteq PSPACE \subseteq EXP$ and we now ask the question if any of this inclusions is strict.

Definition 2.7.1. *For a function $f : \mathbb{N} \to \mathbb{N}$ define*

$$\begin{aligned}\text{NSpace}(f(n)) \quad := \quad & \{L \subseteq \{0,1\}^* \mid \exists \textit{ a non-deterministic turing} \\ & \textit{machine deciding } L \textit{ in } O(f(n)) \textit{ space}\}\end{aligned}$$

and

$$NPSPACE := \bigcup_{c \in \mathbb{N}} \text{NSpace}(n^c)$$

Definition 2.7.2.

$$L := \text{Space}(\log(n))$$

$$NL := \text{NSpace}(\log(n))$$

(of course we mean the work-tape)

If your working-space is $log(n)$ the number of distinct configurations is just $2^{\log(n)}$ so $L \subseteq P$.

Example 2.7.3.

$$MULT = \{(m, n, mn) \mid m, n \in \mathbb{Z}\} \in L$$

We will see that

$$L \subseteq NL \subseteq P$$

Exercise 2.7.4. *Show that:*

a) $\mathrm{Dtime}(f(n)) \subseteq \mathrm{Space}(f(n)) \subseteq \mathrm{NSpace}(f(n)) \subseteq \mathrm{Dtime}(2^{O(f(n))})$

b) $NP \subseteq PSPACE$

2.8 PSPACE-completeness

Definition 2.8.1. *B is* <u>*PSAPCE-complete*</u> *if* $B \in PSPACE$ & $\forall A \in PSPACE :$ $A \leq_P B$.

Definition 2.8.2.

$$TQBF := \{y = Q_1 x_1 \cdots Q_n x_n \mid Q_1, \ldots, Q_n \in \{\forall, \exists\}, \phi \text{ is a boolean formula s.t. } y \text{ is true}\}$$

Example 2.8.3. *for a* <u>*Quantified Boolean Formular*</u> $\forall x_1 \exists x_2 \forall x_3 : \phi(x_1, x_2, x_3)$ *where* ϕ *is a boolean formula and* $x_1, x_2, x_3 \in \{0, 1\}$

Lemma 2.8.4.

$$TQBF \in PSPACE$$

<u>Proof.</u> Let $\psi := Q_1 x_1 \cdots Q_n x_n \phi(x_1, \ldots, x_n)$ be the input QBF with $|\phi| =: m$ where ϕ is given in a format where it can be evaluated in $O(m)$ time. We give the following recursive algorithm to check whether ψ is true: Define $q = \vee$ if $Q_1 = \exists$ and $q = \&$ if $Q_1 = \forall$. Then ψ is true iff

$$Q_2 x_2 \cdots Q_n x_n \phi(0, x_2, \ldots x_n) q Q_2 x_2 \cdots Q_n x_n \phi(1, x_2, \ldots x_n)$$

Now let $s_{n,m}$ be the space used by this algorithm, then

$$\begin{array}{rl} & s_{n,m} = s_{n-1,m} + O(n+m) \\ \Rightarrow & s_{n,m} = O(n(m+n)) \\ \Rightarrow & TQBF \in Space(n^2) \subset PSPACE \end{array}$$

□

Exercise 2.8.5. *Show that* $s_{n,m}$ *can be made* $O(n+m)$.

Lemma 2.8.6.

$$\forall A \in PSPACE : A \leq_P TQBF$$

<u>Proof.</u> Say let M be a turing machine deciding A in space $s(n)$ (which is a polynomial in n). We will construct a QBF $\psi_{M,x}$ of size $(s(n))^2$ whose truth depends on M accepting x. By the proof of the hardness of SAT we have a formula $\phi_{M,x}(c, c')$ that is true iff $c \mapsto c'$ is a valid transition step and it is of size $O(|c| + |c'|) = O(s(n))$. M will take $2^{ds(n)}$ configuration steps to accpet x with $n = |x|$, because $2^{ds(n)}$ is the number of distinct configurations of M on x. Let $\psi_i(c, c')$ be a formula which is true iff $\exists$ at most 2^i steps from $c \mapsto c'$. Then $\psi_{M,x} = \psi_{ds(n)}(c_{start}, c_{end})$ & $\psi_i(c, c') = \exists c''(\psi_{i-1}(c, c'') \,\&\, \psi_{i-1}(c'', c'))$. We need to improve the recurrence:

$$\begin{array}{rcl} \psi_i(c, c') & = & \exists c''(\psi_{i-1}(c, c'') \,\&\, \psi_{i-1}(c'', c')) \\ & = & \exists c'' \forall D_1, D_2((D_1 = c \,\&\, D_2 = c'') \vee (D_1 = c'' \,\&\, D_2 = c') \Rightarrow \phi_{i-1}(D_1, D_2)) \end{array}$$

Remark 2.8.7. *Remember that*

$$\begin{array}{rcl} A \Rightarrow B & \equiv & \not A \vee B \\ \not(A \vee B) & \equiv & \not A \ \& \ \not B \\ A \vee (\exists x : B) & \equiv & \exists x : (A \vee B) \textit{ if } A \textit{ is free of } x \end{array}$$

$\Rightarrow \psi_{ds(n)}(c, c')$ is now a QBF of size $O(s(n)) \cdot ds(n) = O((s(n))^2)$. Observe that $x \in A$ iff $\psi_{ds(n)}(c_{start,x}, c_{stop,x})$ is true which implies that $A \leq_P TQBF$. □

Theorem 2.8.8. *$TQBF$ is $PSPACE$-complete.*

Proof. 2.8.4 and 2.8.6. □

Theorem 2.8.9. *(Savitch [1970])*

$$\text{NSpace}(s(n)) \subseteq \text{Space}((s(n))^2) \quad s(n) \in \Omega(\log(n))$$

Proof. Leet M be a nontederministic turing machine deciding A in Space$(cs(n))$. Then by 2.8.6 for all possible $x \in \{0,1\}^*$: $\psi_{M,x} = \psi_{ds(n)}(c_{start,x}, c_{end,x})$ is of size $O((s(n))^2)$. Using 2.8.4 check whether the QBF $\psi_{M,x}$ is true or not, in space $O((s(n))^2)$. So $x \in A$ can be checked in Space$((s(n))^2)$. □

Corrolar 2.8.10.

$$NPSPACE = PSPACE$$

2.9 NL-completeness

We need a notion of reductioins weaker than P (as $NL \subseteq P$):

Definition 2.9.1.

- *We call $f : \{0,1\}^* \to \{0,1\}^*$ implicitly L-computable if:*

$$L_f := \{< x, i >| \ (f(x))_i = 1\} \in L$$

(here $(f(x))_i$ means the i-th bit of $f(x)$)

$$L'_f := \{< x, i >| \ i \leq |f(x)|\} \in L$$

- *We say $Aleq_L B$ if $\exists$ an impicitly L-computable f s.t. $\forall x \in \{0,1\}^*, x \in A \Leftrightarrow f(x) \in B$.*
- *B is NL-complete if $B \in NL \ \& \ \forall A \in NL : A \leq_L B$.*

Exercise 2.9.2. *a)* $A \leq_L B \leq_L C \Rightarrow A \leq_L C$

b) $A \leq_L B \ \& \ B \in L \Rightarrow A \in L$

c) If B is NL-hard $\&$ $B \in L \Rightarrow NL = L$

Definition 2.9.3.

$$\begin{array}{rl} Path := \{(G, s, t) \mid & G \textit{ directed graph}, s, t \in V(G) \\ & \textit{s.t. there is a directed path from to } t\} \end{array}$$

Lemma 2.9.4.

$$Path \in N$$

Proof. Let G be a directed graph with vertices s and t. Define a non-deterministic turing machine M s.t. it is always at some vertex and a transition step just goes to one of the many possible neighbours.

Exercise 2.9.5. *add the implementation details to this proof to finish it.*

As each vertex can be uniquely refered in $\log(n)$ bits, M can be shown to be using $O(\log(n))$ space. □

Lemma 2.9.6.

$$\forall A \in NL : A \leq_L Path$$

Proof. Let M decide A. For an $x \in A$ construct a graph G where the vertices are all configurations of M, the edges are (c, c') where $c \mapsto c'$ is a valid step of M and $s = c_{start,x}, t = c_{end,x}$

Exercise 2.9.7. *Show that this is a L-reduction.*

□

Theorem 2.9.8. *Path is NL-complete.*

Proof. 2.9.4 and 2.9.6. □

2.10 The Polynomial Hierarchy

We will define an infinite family of complexity-classes, the Polynomial Hirarchy PH, which will basically consist of $NP, NP^{NP}, NP^{NP^{NP}}, \ldots$, will be a generalization of NP and $coNP$ and tries to devide NP from $PSPACE$:

$$NP \subseteq PH \subseteq PSPACE$$

Definition 2.10.1.

$$\begin{aligned} MinDNF := \quad & \{\phi \mid \phi \textit{ is a DNF formula not equivalent to any smaller DNF}\} \\ & \{\phi \mid \forall \ DNF\ \psi, |\psi| < |\phi|, \exists s, \psi(s) \neq \phi(s)\} \end{aligned}$$

This problem does not seem to be in NP (because of $\exists$) and $coNP$ (because of $\forall$), but it is in $PSPACE$. So it's motivating the following definition of a complexity class:

Definition 2.10.2. *A language $\underline{L \in \Pi_2^P}$ iff there is polynomial-time turing-machine M and a constant $C > 0$ s.t. for any string $x \in \{0,1\}^* : x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{|x|^c}, \exists u_2 \in \{0,1\}^{|x|^c}, M(x, u_1, u_2) = 1$.*

Remark 2.10.3. *If you replace $\forall$ by $\exists$ the definition yields the definition of NP and if you replace $\exists$ by $\forall$ the definition of $coNP$. If you exchange $\exists$ and $\forall$ it is unknown which complexity class we will get (since it is not known if $NP \neq coNP$).*

Exercise 2.10.4. *a) $MinDNF \in \Pi_2^P$*

b) $NP, coNP \subseteq \Pi_2^P \subseteq PSPACE$

Definition 2.10.5. *A language* $\underline{L \in \Sigma_2^P}$ *iff there is a polynomial-time turing-machine* M *and a constant* $c > 0$ *s.t.* $\forall x \in \{0,1\}^* : x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{|x|^c}, \forall u_2 \in \{0,1\}^{|x|^c}, M(x, u_1, u_2) = 1$

Exercise 2.10.6. *a)* $NP, coNP \subseteq \Sigma_2^P \subseteq PSPACE$

b) $\Sigma_2^P = co\Pi_2^P$, $\Pi_2^P = co\Sigma_2^P$

c) $NP, coNP \subseteq \Sigma_2^P \cap \Pi_2^P$

Now we define Σ_i^P and Π_i^P by alternating the quantitifiers $\forall$ and $\exists$ for $(i-1)$ times:

Definition 2.10.7.
- *For* $i \geq 1$ *a language* $L \in \Sigma_i^P$ *iff there is a polynomial-time turing-machine* M *and a constant* $c > 0$ *s.t.* $\forall x \in \{0,1\}^*, x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{|x|^c}, \forall u_1 \in \{0,1\}^{|x|^c}, \ldots, Q_i u_i, M(x, u_1, \ldots, u_i) = 1$ *where* $Q_i \in \{\forall, \exists\}$.
- *Define* Π_i^P *analog.*
- $\Sigma_0^P := \Pi_0^P := P$
- $PH := \bigcup_{i \geq 0} \Sigma_i^P$

Exercise 2.10.8. *a)* $\Sigma_1^P = NP$, $\Pi_1^P = coNP$

b) $\forall i \geq 0 : \Sigma_i^P \subseteq \Sigma_{i+1}^P \;\&\; \Pi_i^P \subseteq \Pi_{i+1}^P$

c) $\forall i \geq 0 : \Sigma_i^P = co\Pi_i^P$

d) $\forall i \geq 0 : \Pi_i^P \subseteq \Sigma_{i+1}^P \;\&\; \Sigma_i^P \subseteq \Pi_{i+1}^P$

e) $\forall i \geq 0 : \Sigma_i^P, \Pi_i^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$

f) $PH = \bigcup_{i \geq 0} \Pi_i^P$

g) $PH \subseteq PSPACE$

It is open if $\exists i \geq 0$ s.t.
$$\Sigma_i^P \neq \Sigma_{i+1}^P$$
Even for $i = 0$ it is very hard: $P = \Sigma_0^P \neq \Sigma_1^P = NP$. Just like we believe $P \neq NP \neq coNP$, we also believe the above.

Definition 2.10.9. *If* PH *turns out to be equal to* Σ_i^P *for a constant* i *then we say that* *<u>PH collapses to the i-th level</u>. The* *<u>PH conjecture</u>* *is:* PH *does not collapse.*

The following resulsts follow by the PH-conjecture:

Theorem 2.10.10. $\exists i \geq 1 : \Sigma_i^P = \Pi_i^P \Rightarrow PH = \Sigma_i^P$

<u>Proof.</u> $L \in \Sigma_{i+1}^P$ iff there is a polynomial-time turing-machine M and a constant $c > 0$ s.t. $x \in L$ iff $\exists u_1 \forall u_2 \ldots Q_{i+1} u_{i+1} : M(x, u_1, \ldots, u_{i+1}) = 1$ (1). Now define a related language
$$L' := \left\{(y, z) \mid \forall u_2 \in \{0,1\}^{|y|^c} \exists u_3 \ldots Q_{i+1} u_{i+1} : M(y, z, u_2, \ldots u_{i+1})\right\}$$

clearly: $L' \in \Pi_i^P = \Sigma_i^P \Rightarrow \forall (y,z), (y,z) \in L'$ iff $\exists v_1 \forall v_2 \dots Q_i v_i : M'(y,z,v_1,\dots,v_i) = 1 \quad (2)$. Now (1) can be rewritten as:

$$x \in L \Leftrightarrow \exists u_1, (x,u_1) \in L'$$

. Which by (2) gives iff $\exists u_1 \exists v_1 \exists v_2 \forall v_2 \dots Q_i v_i M'(x,u_1,v_1,\dots,v_i) = 1$. $L \in \Sigma_i^P \Rightarrow \Sigma_{i+1}^P = \Sigma_i^P \Rightarrow \Pi_{i+1}^P = \Pi_i^P$ (by applying *co*− on both sides). so $\Sigma_{i+1}^P = \Sigma_i^P = \Pi_i^P = \Pi_{i+1}^P$. Now we can repeat the argument again and again. □

Corrolar 2.10.11.

$$NP = coNP \Rightarrow PH = NP$$

Theorem 2.10.12.

$$\exists i \geq 0 : \Sigma_i^P = \Sigma_{i+1}^P \Rightarrow PH = \Sigma_i^P$$

Proof. Applying *co*− to both sides of $\Sigma_i^P = \Sigma_{i+1}^P$ gives $\Pi_i^P = \Pi_{i+1}^P$. As $\Pi_i^P \subseteq \Sigma_{i+1}^P$ & $\Sigma_i^P \subseteq \Pi_{i+1}^P$ we get $\Sigma_{i+1}^P = \Sigma_i^P = \Pi_i^P = \Pi_{i+1}^P$. Applying 2.10.10 on $\Sigma_{i+1}^P = \Pi_{i+1}^P$ we get $PH = \Sigma_{i+1}^P = \Sigma_i^P$. □

Corrolar 2.10.13.

$$P = NP \Rightarrow PH = P$$

Suppose A is PH-complete then $A \in \Sigma_i^P$ for some $i \geq 0$. Which implies that $PH = \Sigma_i^P$. Thus the PH-conjecture implies that A cannot exist. And as a

Corrolar 2.10.14. *PH-cojecture $\Rightarrow PH \subsetneq PSPACE$.*

2.11 Σ_i^P-complete problems

Nevertheless there is no PH-complete, we can define Σ_i^P-complete problems for all constants $i \in \mathbb{N}$:

Definition 2.11.1. *For $i \geq 1$, define*

$$\begin{aligned}\Sigma_i SAT := \quad & \{\phi(u_1,\dots,u_i) \mid \phi(x_1,\dots,x_n) \\ & \textit{ is a boolean formular with a partition of} \\ & x_1,\dots,x_n \textit{ into } u_1,\dots,u_i \textit{ s.t.} \\ & \exists u_1 \forall u_2 \dots Q_i u_i\, [\phi(u_1,\dots,u_i) = 1]\}\end{aligned}$$

For exmplate u_1 could be $x_1x_2x_3$ and $u_2 = x_2x_6$ etc.

Theorem 2.11.2. *$\Sigma_i SAT$ is Σ_i^P-complete, for every $i \in \mathbb{N}^{\geq 1}$.*

Proof. First we see by the definition of $\Sigma_i SAT$ that it is contained in Σ_i^P, you can take a turing machine that evaluates $\phi(x_1,\dots,x_n)$ for a given valuation $u_1,\dots,u_i$. $\phi \in \Sigma_i SAT$ iff $\exists u_1 \forall u_2 \dots Q_i u_i [M(\phi,u_1,\dots,u_i) = 1]$. Now we will show that $\Sigma_i SAT$ is Σ_i^P-hard: Any $L \in \Sigma_i^P$ has a corresponding polynomial-time turing-machine M by definition. The computation of M on $(x,y_1,\dots,y_i)$ can be captured in a boolean formula $\phi_{x,y_1,\dots,y_i}$ (as in the proof of NP-hardness of SAT). Thus $L \leq_P \Sigma_i SAT$. □

Definition 2.11.3. $\Pi_i SAT$:= *flip the quantifiers in the definition of* $\Sigma_i SAT$.

Theorem 2.11.4. *$\Pi_i SAT$ is Π_i^P-complete.*

2.12 PH via oracle machines

Definition 2.12.1. *For complexity classes C_1 and C_2, we define*

$$C_1^{C_2} := \bigcup_{L \in C_2} C_1^L$$

For e.g. $NP^{NP} = NP^{SAT}$. We will now see that $\Sigma_2^P = NP^{NP}$, i.e. a non-deterministic turing machine using SAT as an oracle can solve $\Sigma_2 SAT$ in polynomial time. Note that $P^P = P$ but it is an open question whether $NP^{NP} = NP$ or not.

Theorem 2.12.2.

$$\forall i \geq 2 : \Sigma_i^P = NP^{\Sigma_{i-1}^P}$$

Proof. We will demonstrate the proof-idea by prooving that $\Sigma_2^P = NP^{NP}$ which will naturaly generalize to every i.

Suppose that $L \in \Sigma_2^P$ then there exists a polynomial-time turing-machine and a constant $c > 0$ s.t.

$$\forall x \in \{0,1\}^* : x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{|x|^c} \forall u_2 [M(x, u_1, u_2) = 1]$$

The associated language $L' := \left\{(y,z) \mid \forall u_2 \in \{0,1\}^{|y|^c}, M(y,z,u_2) = 1\right\}$. Than we can rewrite the above to $x \in L$ iff $\exists u_1 (x, u_1) \in L'$. As $L' \in coNP \subseteq P^{NP}$ and we get $L \in NP^{NP}$. So $\Sigma_2^P \subseteq NP^{NP}$.

Now suppose that $L \in NP^{NP} = NP^{SAT}$. Say L is decided in polynomial-time by a non-deterministic turing machine N using SAT as an oracle and let $\overline{c} = (c_1, \ldots, c_m) \in \{0,1\}^m$ be the transition-choices of N on x. It queries the oracle on formulas: $\phi_{\overline{c},1}, \ldots, \phi_{\overline{c},k}$ and got answers $\overline{a} = (a_1, \ldots, a_k) \in \{0,1\}^k$. Then

$$\begin{aligned} x \in L \quad & \Leftrightarrow \quad \exists \overline{c}, \overline{a} : (N \text{ accepts } x \text{ on the path } \overline{c}) \ \& \ (\overline{a} \text{ are correct answers}) \\ & \Leftrightarrow \quad \exists \overline{c}, \overline{a}, u_1, \ldots, u_k \forall v_1, \ldots, v_k : (N \text{ accepts } x \text{ on the path } \overline{c}) \ \& \\ & \qquad (\forall 1 \leq i \leq k, (a_i = 0 \Rightarrow \phi_{\overline{c},i}(v_i) = 0) \ \& \ (a_i = 1 \Rightarrow \phi_{\overline{c},i}(u_i) = 1)) \end{aligned}$$

Exercise 2.12.3. *Proof the last equivalence above.*

So $L \in \Sigma_2^P$ and $NP^{NP} \subseteq \Sigma_2^P$. Thus $\Sigma_2^P = NP^{NP}$. □

Exercise 2.12.4. *complete the proof for i > 2 by showing that $\Sigma_i^P = NP^{\Sigma_{i-1}^P SAT}$.*

Corrolar 2.12.5.

$$\Sigma_2^P = NP^{NP}, \Sigma_3^P = NP^{NP^{NP}}, \ldots$$

We have the following situation:

$$P \subseteq NP \subseteq NP^{NP} \subseteq \ldots \subseteq PH \subseteq PSPACE$$

And it is unknown if the last containment is sharp or not, but we will see counting problems which are in between.

2.13 Between PH and $PSPACE$

Definition 2.13.1. *Define a function*

$$\begin{array}{rccl} \#SAT: & \{\phi \mid \phi \text{ is a boolean formula}\} & \to & \mathbb{N} \\ & \phi & \mapsto & \text{number of satisfying assignments of } \phi \end{array}$$

It is an open question if $\#SAT$ is efficiently computable. It can be computed in polynomial space.

Definition 2.13.2.

$$\begin{array}{rcl} FP & := & \{f : \{0,1\}^* \to \{0,1\}^* \mid \\ & & f \text{ is computable by a polynomial-time turing-machine } M_f\} \end{array}$$

It is open if $\#SAT \in FP$ but if it is not, then $P = NP$.

Definition 2.13.3. *A function $f : \{0,1\}^* \to \mathbb{N}$ is in $\#P$ if there exists a polynomial-time turing-machine M and $c \in \mathbb{R}^{>0}$ s.t. for all $x \in \{0,1\}^* : f(x) = |\{y \in \{0,1\}^{|x|^c} \mid M(x,y) = 1\}|$. In other words $\#P$ is the collection of functions that count the number of accepting paths of a polynomial-time non-deterministic turing-machine .*

Exercise 2.13.4. *a) $\#SAT \in \#P$*

b) $NP \subseteq P^{\#SAT} \subseteq P^{\#P}$

c) $P^{\#P} \subseteq EXP, P^{\#P} \subseteq PSPACE$

d) $FP \subseteq \#P$

e) "Counting points on a variety over finite fields" $\in \#P$

It is open to show that $FP \subsetneq \#P$ (arithmetic version of $P \neq NP$)

Remark 2.13.5. *$\#P$ has an analogous decision problems class: PP.*

Definition 2.13.6. *A language $L \in PP$ if there exists a polynomial-time turing-machine M and $c \in \mathbb{R}^{>0}$ s.t. for all $x \in \{0,1\}^*$:*

$$x \in L \Leftrightarrow |\{y \in \{0,1\}^{|x|^c} \mid M(x,y) = 1\}| \geq \frac{1}{2} 2^{|x|^c}$$

In other words PP corresponds to t_0 computing most-signifant-bit of $\#P$-function.

Exercise 2.13.7. *a) $\#SAT \in FP^{PP}$*

b) $P^{\#P} = P^{PP}$

2.14 $\#P$-completeness

Definition 2.14.1. *We call a function $f : \{0,1\}^{\mathbb{N}}$ is $\#P$-comlete if $f \in \#P$ and $\#P \subset FP^f$ (here FP^f is the set of functions computable by a polynomial-time oracle turing-machine M using f as an oraclre, this is called "turing reduction", in contrast to the karp-reduction).*

Exercise 2.14.2. *If f is $\#P$-complete and $f \in FP$ then $\#P = FP$.*

Theorem 2.14.3. *$\#SAT$ is $\#P$-complete*

Proof. Recall that the computation of a polynomial-time non-deterministic turing-machine M can be encoded in a boolean formula. Thus, for any $g \in \#P$ with $g(x)$ equals the number of acceptings paths of M on input x. We get a boolean formula $\phi_{M,x}$ s.t. the number of satisfying assignments of $\phi_{M,x} = g(x)$. So $g \in FP^{\#SAT}$. Since g was arbitrary: $\#P \subseteq FP^{\#SAT}$. □

2.15 $PERMANENT$ and $\#P$

Definition 2.15.1. *Let F be a field, for $A = (A_{ij})_{1 \le i,j \le n} \in \mathbb{F}^{n \times n}$ the permanent is defined as:*

$$\mathrm{per}(A) := \sum_{\sigma \in S_n} A_{1,\sigma(1)} \cdot \ldots \cdot a_{n,\sigma(n)}$$

here S_n is the group of $n!$ permutations of $\{1, \ldots, n\}$. So per *is defined like* det *but without any signs.*

We will now show, that per is $\#P$-complete:

Lemma 2.15.2. per *for* 0-1-*matrices $\in \#P$.*

Proof. Let $A \in \{0,1\}^{n \times n}$, then $\mathrm{per}(A) = |\{\sigma \in S_n \mid \prod_{i=1}^n a_{i,\sigma(i)} = 1\}|$. Define a non-deterministic turing-machine M that on input A, guesses $\sigma \in S_n$ and accpets A iff $\prod_{i=1}^n A_{i,\sigma(i)} = 1$. Hence $\mathrm{per}(A)$ = number of accepting paths of M on input A it follows that per for 0-1-matrices $\in \#P$.

Exercise 2.15.3. *Use the above idea to show:* per *for matrices over a finite field $\in \#P$.*

This shows that per $\in FP^{\#SAT}$. □

Definition 2.15.4. *Given $A \in \mathbb{F}^{n \times n}$ and view it as the adjacency matrix of a weightes digraph on n vertices (with weight-function w and which we will also call A). Then a cycle cover C of the graph A is a subgraph of A having the n vertices and each vertex v has $\delta^+(v) = \delta^-(v) = 1$ (in-degree and out-degree are one). Thus a cycle cover C of A is basically a disjoint union of cycles covering all vertices of A. The weight of C is defined as*

$$wt(C) := \prod_{v \in E(C)} w(v)$$

Exercise 2.15.5. *a) $A_{1,\sigma(1)} \cdot \ldots \cdot A_{n,\sigma(n)}$ equals the weight of the cycle cover in the graph A corresponding to the cycle-decomposition of permutation σ.*

b)

$$\mathrm{per}(A) = \sum_{C \text{ cycle-cover of } A} wt(C)$$

Lemma 2.15.6. per *for* 0-1-*matrices is $\#P$-hard.*

Proof. We will relate $\#3SAT$ with per. Let ϕ be a CNF formula with m clauses (each with exactly 3 literals) $F_1, \ldots, F_m$ and n, named $x_1, \ldots, x_n$, variables. Then we will construct a graph A with sum of its weighted cycle-covers kind of "equals to" the number of satisfying assignments of ϕ:

variables: $x_1, \ldots, x_n \mapsto$ variable gadgets (graphs) $V_1, \ldots, V_n$

clauses: $F_1, \ldots F_m \mapsto$ clause gadgets $C_1, \ldots, C_m$

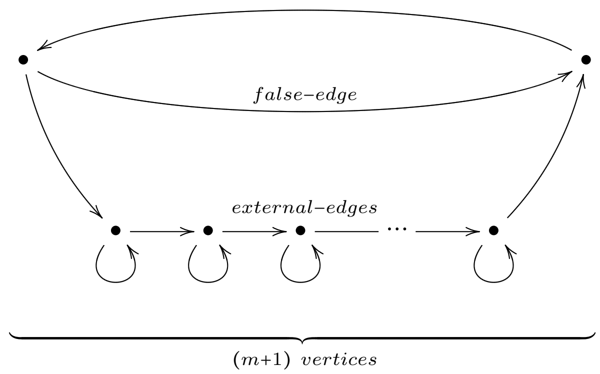
occurrence: x_i / $\overline{x_i}$ occures in $F_j \mapsto XOR_{i,j}$-gadgets

These $n + m + 3m$ gadgets together will form the graph A s.t.

$$\sum_{C \text{ cycle-cover}} wt(C) = 4^{3m} \text{ number of satisfying assignments of } \phi$$

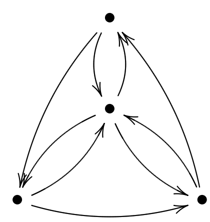
Unlabled edges are understood to have weight 1 in the following.

Each variable x_i is converted to a graph gadget V_i:



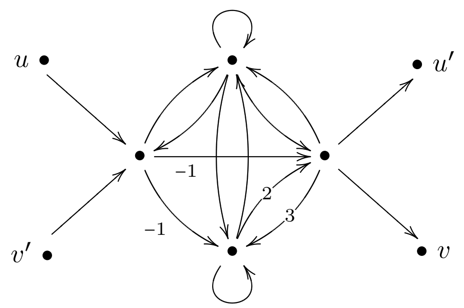
the edges between the bottom $m + 1$ vertices are called external. The cycle cover of V_i using the external edges corresponds to $x_i = T$ and the one using the upper edge (called false edge) corresponds to $x_i = F$.

Each clause F_j is converted to a graph gadged C_j:



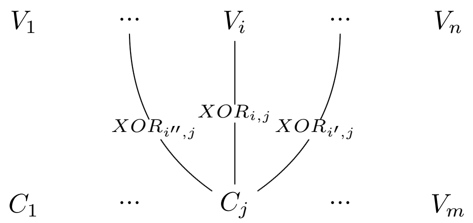
The 3 outer edges (called external again) correspond to the 3 literals of F_j. This graph has 3 cycle-covers each of weight 1 and corresponding to a droped outer (external) edge. The dropped external edge specifies the literal in F_j set to "True".

If a variable x_i (not it's complement $\overline{x_i}$) appears in F_j then we connect the j-th external edge (u, u') of V_i to the x_i-external edge of C_i by a graph gadget $XOR_{i,j}$:

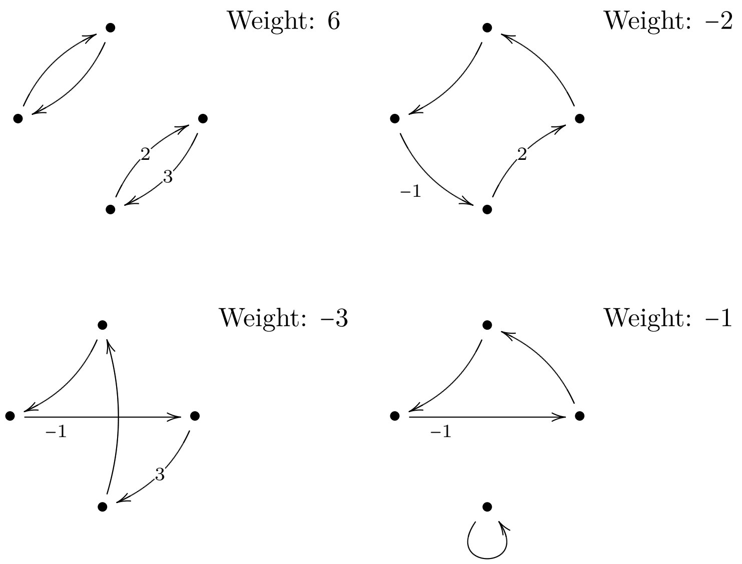


Finally if $\overline{x_i}$ is in F_j then we connect the false edge (u, u') of V_i to the x_i external edge (v, v') of C_j by $XOR_{i,j}$.

We call the resulting graph A' and we can see by the following figure that the number of edges is quadratic in n and m.

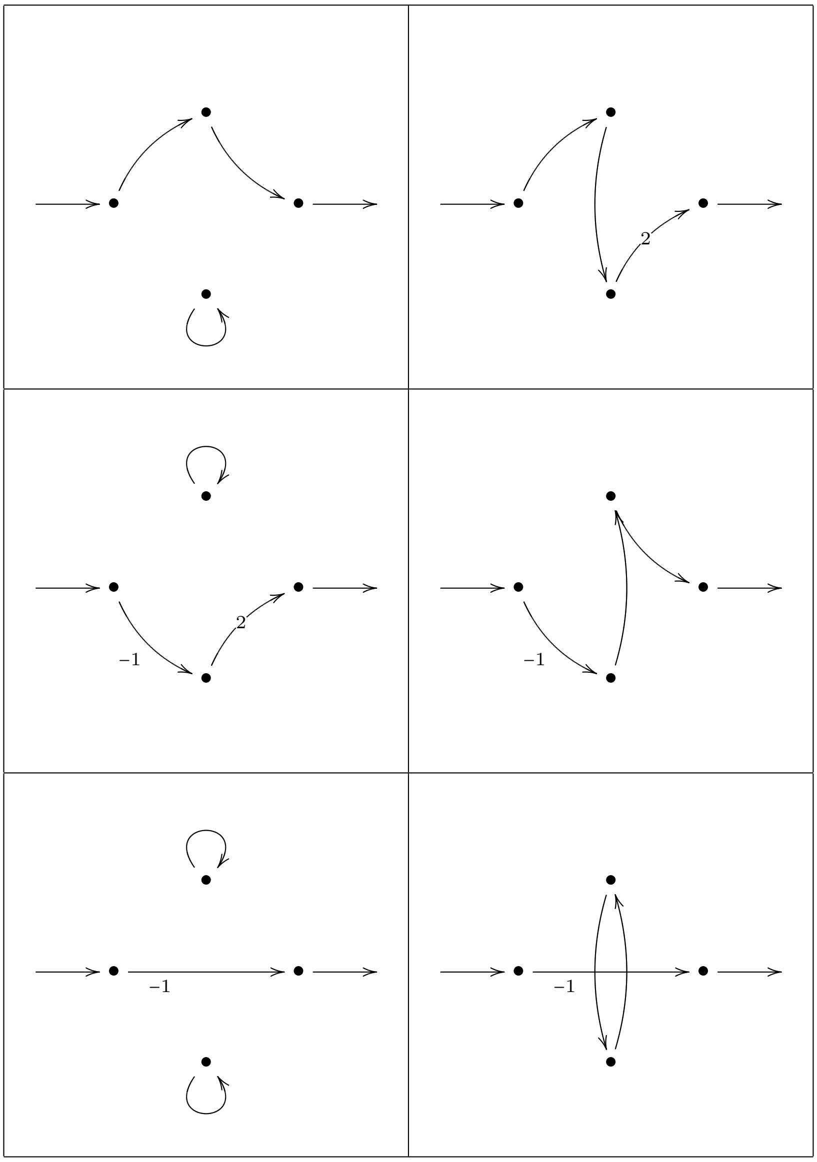


Now observe that the cycle-covers of $XOR_{i,j} - \{u, v, u', v'\}$ sum up to 0:



Thus the cycle-covers of A' that "matter" in $\mathrm{per}(A')$ are the ones for which $XOR_{i,j}$ contributes to exactly one of the paths (u, u') or (v, v') in the cycle-cover (hence the name XOR). Consider such a cycle-cover $\mathcal{C}$ of A': For every C_j there will be an external edge not appearing in $\mathcal{C}$ say x_i. $XOR_{i,j}$ ensures that the j-th external edge in V_i appears in $\mathcal{C}$. Thus the "True" cycle of V_i appears in $\mathcal{C}$ and we can take $x_i = T$. So $\mathcal{C}$ induces a satisfying assignment of ϕ.

The weightes sum of paths from u to u' in $XOR_{i,j}$ is -2:

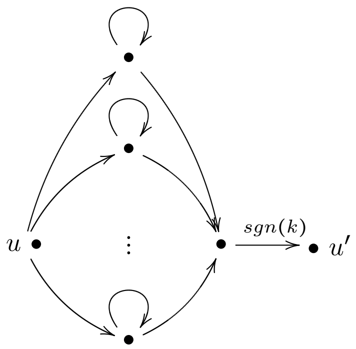


Thus, for a satisfying assignment s of ϕ:

$$\sum_{\mathcal{C} \text{ is a cycle cover of } A' \text{ corrresponding to } s} wt(\mathcal{C}) = (-2)^{3m}$$

$\Rightarrow \operatorname{per}(A') = (-2)^{3m} \#(\phi)$.

The final question is how to reduce A' to a 0-1-matrix A'': First reduce it to a $\{-1, 0, 1\}$-matrix by replacing every k-weighted edge (u, u') withh $k \in \mathbb{Z} - \{0, 1\}$ by the following graph (there are k vertices on top of each other):



Say $A'' \in \{-1,0,1\}^{n\times n}$ then $\mathrm{per}(A'') \in [-(n!), n!] \cap \mathbb{Z}$. So use $N := 2^{n^2}$, work (mod N) and replace -1 by $(N-1)$ in the graph. So we get a $\{0,1,N-1\}$-matrix A s.t.

$$\mathrm{per}(A) \equiv (-2)^{3m} \#(\phi) \pmod N$$

Exercise 2.15.7. *Convert a $\{0,1,N-1\}$-matrix to a $\{0,1\}$-matrix with the same permanent.*

$$\Rightarrow \#SAT \in FP^{\text{permanent of 0-1-matrix}}$$

□

Theorem 2.15.8. *(Valiant 1979) PERMANENT for 0-1-matrix is #P-complete.*

Theorem 2.15.9. *(Toda, 1991)*

$$PH \subseteq P^{\#P}$$

First remember that $PH = \bigcup_{i=0} \Sigma_i^P$ and $P^{\#P} = P^{\#SAT}$

We will use some probabilistic methods during this proof.

Idea: We give a new type of reduction (i.e. random reduction) from any Σ_i^P to a new class $\oplus P$ (parity-p). Surprisingly, this "randomized reduction" will help in proving a deterministic statement.

Definition 2.15.10. *A language $L \in \oplus P$ (called parity-P) if there is a polynomial-time non-deterministic turing-machine M s.t. for all $x \in \{0,1\}^*$: $x \in L$ iff #(accepting paths of M on x) is odd.*

Definition 2.15.11.

$$\oplus SAT := \{\phi \mid \phi \text{ is a boolean formula that has an odd number of satisfying assignments}\}$$

Exercise 2.15.12. *$\oplus SAT$ is $\oplus P$-complete.*

Open question: $\oplus P = P \Rightarrow NP = P$

The following theorem gives an evidence towards that:

Theorem 2.15.13. *(Valiant-Vazisani) There is a polynomial-time turing-machine A s.t. for any boolean formula ϕ on n variables:*

$$\begin{array}{lll} \phi \in SAT & \Rightarrow & \Pr_r\left[A(r,\phi) \in \oplus SAT\right] \geq \frac{1}{8n} \\ \phi \notin SAT & \Rightarrow & \Pr_r\left[A(r,\phi) \in \oplus SAT\right] = 0 \end{array}$$

Remark 2.15.14. *This we state as "SAT randomly reduces to $\oplus SAT$"*

Idea: Given a CNF-boolean formula ϕ we want to ransform ϕ into a fomula ϕ' that has 0 or 1 satisfying assignments depending on whether ϕ is unsatisfiable respectively. To describe this transformation we need the following lemma "hash functions":

Lemma 2.15.15. *For a matrix $B \in \{0,1\}^{k\times n}$ and a vector $b \in \{0,1\}^k$, consider the transformation:*

$$\begin{array}{lllll} h_{B,b} & : & \{0,1\}^n & \to & \{0,1\}^k \\ & & x & \mapsto & (Bx+b) \pmod 2 \end{array}$$

a) For a fixed $x \in \{0,1\}^n : \Pr_{B,b}\left[h_{B,b}(x) = 0^k\right] = 2^{-k}$

b) For a fixed $x \neq x' \in \{0,1\}^n : \Pr_{B,b}\left[h_{B,b}(x) = h_{B,b}(x') = 0^k\right] = 2^{-k}$

c) Let $S \subseteq \{0,1\}^*$ s.t. $2^{k-2} \leq |S| \leq 2^{k-1}$.

Then

$$\Pr_{B,b}\left[\#\{x \in S \mid h_{B,b}(x) = 0^k\} = 1\right] \geq \frac{1}{8}$$

Let us first use the lemma to proof the theorem:

Proof. Description of A: Given a CNF ϕ on n variables randomly pick $k \in \{2,\dots,n+1\}, B \in \{0,1\}^{k\times n}$ and $b \in \{0,1\}^k$. Output the boolean formula:

$$\psi(x) := \phi(x) \ \& \ \left[h_{B,b}(x) = 0^k\right]$$

The last term is expressed as a boolean formula.

Note that: If ϕ is unsatisfiable then ψ has 0, hence even, satisfiying assignments. If ϕ is satisfiable then, let $S := \{x \in \{0,1\}^n \mid \phi(x) = 1\}$ with probability $\geq \frac{1}{n}$ we whould have chosen k s.t. $2^{k-2} \leq \#S \leq 2^{k-1}$.

Conditioned on this, with probability $\geq \frac{1}{8}$ we have chosen (B,b) s.t.

$$\#\{x \in S \mid h_{B,b}(x) = 0^k\} = 1$$

Thus with probability $\geq \frac{1}{8n}$ we whould have chosen (k,B,b) s.t.

$$\#(\text{satisfying assignmen of } \psi) = 1$$

hence odd. □

Proof. Let us now proof 2.15.15.

a) $h_{B,b}(x) = 0^k$, $Bx = -b \pmod 2$. If we first pick B then the probability of picking $b \equiv (-Bx) \pmod 2$ is $\frac{1}{2^k}$ so $\Pr_{B,b}[h_{B,b}(x) = 0^k] = 2^{-k}$

b)

$$\begin{aligned}\Pr_{B,b}[Bx=-b\ \&\ Bx'=-b] &= \Pr_{B,b}[Bx=-b]\cdot\Pr_{B,b}[Bx'=-b\mid Bx=-b]\\ &= 2^{-k}\cdot\Pr_{B,b}[B(x'-x)=0\mid Bx=-b]\\ &= 2^{-k}\Pr_{B}[B(x'-x)=0]=2^{-2k}\end{aligned}$$

c) Let N be the random variable $\#\{x\in S\mid h_{B,b}(x)=0^k\}$. Then by inclusion-exclusion:

$$\begin{aligned}\Pr_{B,b}[N\geq 1] &\geq \left(\sum_{x\in S}\Pr_{B,b}[h_{B,b}(x)=0^k]\right)-\left(\sum_{x<x'\in S}\Pr_{B,b}[h_{B,b}(x)=h_{B,b}(x')=0^k]\right)\\ &= |S|\cdot 2^{-k}-\binom{|S|}{2}\cdot 2^{-2k}\end{aligned}$$

similarly:

$$\begin{aligned}\Pr_{B,b}(N\geq 2) &\leq \sum_{x<x'\in S}\Pr_{B,b}[h_{B,b}(x)=h_{B,b}(x')=0^k]\\ &= \binom{|S|}{2}\cdot 2^{-2k}\end{aligned}$$

So

$$\begin{aligned}\Pr_{B,b}[N=1] &= \Pr_{B,b}[N\geq 1]-\Pr_{B,b}[N\geq 2]\\ &\geq |S|2^{-k}-2\binom{|S|}{2}2^{-2k}\\ &= |S|2^{-k}-|S|(|S|-1)2^{-2k}\\ &\geq |S|2^{-k}-(|S|2^{-k})^2\\ &\geq |S|2^{-k}(1-|S|2^{-k})\\ &\geq \tfrac{1}{4}(1-\tfrac{1}{2})=\tfrac{1}{8}\end{aligned}$$

□

Now we will use the idea in the proof that NP randomly reduces to $\oplus P$ repeatedly to show: PH randomly reduces to $\oplus P$. We will replace $\forall$ and $\exists$ by a $\oplus$ quantifier one-by-one.

Definition 2.15.16. *For any boolean formula $\phi(\overline{z})$, $\psi=\oplus\overline{z}\phi(\overline{z})$ is true iff # satisfying assignments of $\phi(\overline{z})$ is odd. See this as a definition of the quantifier $\oplus$.*

Lemma 2.15.17. *Let $x\in\mathbb{N}$ be a constant. There is a deterministic polynomial time transformation A s.t. for any formula ψ starting with c alternating $\forall/\exists$-quantifiers:*

$$\begin{aligned}&\textit{If } \psi \textit{ is true} &&\Rightarrow \Pr_r\left[A(r,\psi)\in\oplus SAT\right]=\tfrac{2}{3}\\ &\textit{If } \psi \textit{ is false} &&\Rightarrow \Pr_r\left[A(r,\psi)\in\oplus SAT\right]=0\end{aligned}$$

Proof. Let ψ be a formula with $c'\leq c$ alternating $\forall/\exists$-quantifiers. As our aim is to replace them one-by-one with the $\oplus$-quantifier, let us assume that ψ starts with a $\oplus$-quantifier. We will demonstrate the proof for $\psi=\oplus z\in\{0,1\}^l\,\exists x\in\{0,1\}^n,\forall w\in\{0,1\}^k,\phi(z,x,w)$ with a boolean formula ϕ using the $l+n+k$ variables. By the proof of 2.15.13 there exists a formula τ s.t. for a random string r (we will drop the domains of x,w and w in the following)

$$\Pr_r\left[\oplus x\forall w\left(\phi(z,x,w)\ \&\ \tau(x,r)\right)\text{ is true}\right]\geq\frac{1}{8n}$$

if $\exists x\forall w\phi(z,x,w)$ is true .

$$\Pr_r\left[\oplus x\forall w\left(\phi(z,x,w)\ \&\ \tau(x,r)\right)\text{ is true}\right]=0$$

if $\exists x \forall w \phi(z,x,w)$ is false

So $\Pr_r\left[\oplus z \oplus x \forall w\left(\phi(z,x,w) \;\&\; \tau(x,r)\right) \text{ is true}\right] \geq \left(\frac{1}{8n}\right)^2$ if ψ is true.

So we have randomly reduced ψ to $\oplus z \oplus x \forall w\left(\phi(z,x,w) \;\&\; \tau(x,r)\right)$ but the probability of success is extremly low. We improve it by first boosting equation 2.15. For any z, repeat the transformation in equation 2.15 for $t \in \mathbb{N}$ random strings $r_1, \ldots, r_t$.

$$p := \Pr_{r_1,\ldots,r_t}\left[\bigvee_{i=1}^{t} \oplus x \forall w \phi(z,x,w) \;\&\; \tau(x,r_i) \text{ is true}\right]$$

- if $\exists x \forall w \phi(z,x,w)$ is true then $p = 1 - (1 - \frac{1}{8n})^t$
- if $\exists x \forall w \phi(z,x,w)$ is false then $p = 0$

Consider all $z \in \{0,1\}^l$:

$$p := \Pr_{r_1,\ldots,r_t}\left[\oplus z \bigvee \oplus x \forall w \phi(z,x,w) \;\&\; \tau(x,r_i) \text{ is true}\right]$$

Exercise 2.15.18. *a) if ψ is true then $p = 1 - 2^l\left(1 - \frac{1}{8n}\right)^t$*

b) if ψ is false then $p = 0$

Exercise 2.15.19. *Proof that $1 - 2^l\left(1 - \frac{1}{8n}\right)^t > \frac{2}{3}$*
Hint: Pick $t = 16n(l+1)$ then something like

$$2^l\left(1 - \frac{1}{8n}\right)^t < 2^l\left(\left(1 - \frac{1}{8n}\right)^{8n}\right)^{2l} \leq 2^l\left(\frac{1}{2n}\right)^{2l} = \frac{1}{2^l}$$

happens

We have randomly reduced $\psi = \oplus z \exists x \forall w \phi(z,x,w)$ to $\psi' = \oplus z \bigvee_{i=1}^{t} \oplus x \forall w \phi'(z,x,w,r_i)$. How to remove $\vee$-operators? We develope some usefull quantified boolean formula algebra (for a boolean formla F denote the number of satisfying assignments as $\#F$):

a) For boolean formulas $F(\overline{x})$ and $G(\overline{x})$ we define

$$(F+G)(\overline{x},u) := ((u=0) \;\&\; F(\overline{x})) \vee (u = 1 \;\&\; G(\overline{x}))$$

Observation: $\#F = \#G = \#(F+G)$

b) For boolean formula $F(\overline{x})$ and $G(\overline{y})$ define:

$$(F \cdot G)(\overline{x},\overline{y}) := F(\overline{x}) \;\&\; G(\overline{y})$$

Observation: $\#(F \cdot G) = (\#F) \cdot (\#G)$

c) For a boolean formla $F(\overline{x})$ we define

$$(F+1)(\overline{x},u) := (u = 0 \;\&\; F(\overline{x})) \vee (u = 1 \;\&\; \overline{x} = 0^n)$$

Observation: $\#(F+1) = \#F + 1$

d) $\oplus \overline{x} F_1(\overline{x}) \vee \oplus \overline{y} F_2(\overline{y}) = \oplus(\overline{x},\overline{y},u_1,u_2,u_3)((F_1+1)(\overline{x},u_1) \cdot (F_2+1)(\overline{y},u_2) + 1)(\overline{x},\overline{y},u_1,u_2,u_3)$

Thus in this way we can randomly reduce $\psi = \oplus z \exists x \forall w \phi(z, x, w)$ to

$$\psi'' = \oplus z \oplus x^* \forall w \phi''(z, x^*, w)$$

Next we can remove the $\forall$-quantifier because:

Exercise 2.15.20.

$$\oplus \overline{x} \forall \overline{y} F(\overline{x}, \overline{y}) = \oplus \overline{x} \exists \overline{y} \not{F} (\overline{x}, \overline{y})$$

Then repeat the older steps on $\oplus z \oplus x^* \exists w \not{\phi}''$ and end up with

$$\oplus (z, x^*, w^*) \phi'''(z, x^*, y^*)$$

and because the number of quantifiers in ψ is constant we only get a polynomial blowup of ϕ''' in the size of ϕ and thee probability of $\frac{2}{3}$ can be achieved. So PH randomly reduces to $\oplus P$ (with a "decent" probybility of $\frac{2}{3}$). □

As $\oplus P \subset P^{\#P}$ we already have a randomized reduction from PH to $\#P$. How to derandomize it?

For that we show a proberty of $\oplus P$:

Lemma 2.15.21. *Let ψ be a boolean formula and $m \in \mathbb{N}_0$ then there exists a deterministic polynomial-time transformation T s.t. $\phi := T(\psi, 1^m)$ is a boolean formula s.t.*

$$\begin{array}{lcl} \#\psi = 1 \pmod 2 & \Rightarrow & \#\phi \equiv -1 \pmod{2^{m+1}} \\ \#\psi = 0 \pmod 2 & \Rightarrow & \#\phi \equiv 0 \pmod{2^{m+1}} \end{array}$$

Proof. We build ϕ iteratively. Let $\phi_0 := \psi$. Consider $\psi_i := 4\phi_0^3 + 3\phi_0^4$.

$$\begin{array}{lcl} \#\phi_0 = 1 \pmod 2 & \Rightarrow & \#\phi_1 = -1 \pmod{2^2} \\ \#\phi_0 = 0 \pmod 2 & \Rightarrow & \#\phi_1 = 0 \pmod{2^2} \end{array}$$

Exercise 2.15.22. *The recurrence $\phi_{i+1} = 4\phi_i^3 + 3\phi_i^4$ gives*

$$\begin{array}{lcl} \#\phi_0 = 1 \pmod 2 & \Rightarrow & \#\phi i + 1 = -1 \pmod{2^{2^{i+1}}} \\ \#\phi_0 = 0 \pmod 2 & \Rightarrow & \#\phi i + 1 = 0 \pmod{2^{2^{i+1}}} \end{array}$$

□

Proof. (of theorem 2.15.9) Let $L \in PH$. We wil show how to check $x \in$ using $\#SAT$ as an oracle. By the last two lemmas there exists a deterministic polynomial-time turing-machine M and $m = poly(|x|)$ s.t.

$$\begin{array}{lcl} x \in L & \Rightarrow & \Pr_{r \in \{0,1\}^m}[\#acceptingpaths(M(x, r)) \equiv -1 \pmod{()2^{m+1}}] \\ x \notin L & \Rightarrow & \forall r \in \{0,1\}^m \#acceptingpaths(M(x, r)) \equiv 0 \pmod{()2^{m+1}} \end{array}$$

Remark 2.15.23. *M is the non-deterministic turing-machine corresponding to ϕ in the last lemma i.e. M just guesses a satisfying assignment of ϕ and then verifies it.*

Now define a polynomial-time non-deterministic turing-machine M' that on input x, guesses $r \in \{0,1\}^m$ and accepts iff M accepts (x, r).

$$\#\text{ accepting paths of } (M'(x)) = \sum_{r \in \{0,1\}^m} \#\text{accepting paths}(M(x, r))4$$

If $x \in L$ then #accepting paths$(M'(x)) \pmod{2^{m+1}} \in [-2^m, -\frac{2}{3}2^m]$.

If $x \notin L$ then #accepting paths$(M'(x)) \pmod{2^{m+1}}$ is 0.

Thus, computing #accepting paths$(M'(x))$ is anough to solve L which means that $PH \subseteq P^{\#P}$. □

2.16 Probabilistic turing machines

Now we will formalize randomized computation. Randomness has been useful in the last four decades to:

a) prooving theorems in complexity whose statements did not call for randomness

b) develop simpler and faste algorithms for several problems in combinatorial optimization (simplex, quicksort), algebraic computation, machine learning and network routing.

Definition 2.16.1. *We call M a probabilistic turing machines (PTM) if M has two transition functions δ_0 and δ_1 and in each transition step M randomly follows δ_0 or δ_1 eaach with probability $\frac{1}{2}$. We say that a probabilistic turing-machine M decides L if for any $x \in \{0,1\}^*$ $x \in L \Leftrightarrow \Pr_{steps}[M \text{ accepts } x]\frac{2}{3}$*

Definition 2.16.2. *For a function $T : \mathbb{N} \to \mathbb{N}$ a probabilistic turing-machine M decides L in time $T(n)$ if M decides L and for all $x \in \{0,1\}^*$: M halts on x in $\leq T(|x|)$ steps regardless of its random choices.*

$$\mathrm{BPTime}(T(n)) := \{L \subseteq \{0,1\}^* \mid \textit{some probabilistic turing-machine } M \textit{ decides } L \textit{ in time } O(T(n))\}$$

(bounded probabilistic)

$$BPP := \bigcup_{c\in\mathbb{N}} \mathrm{BPTime}(n^c)$$

Exercise 2.16.3. $P \subseteq BPP \subseteq PSPACE \subseteq EXP$

It is open if $P = BPP$ (but "evidence" suggests that it is) or if $BPP = NP$ (we will connect with PH collapse).

Exercise 2.16.4. *(Alternative definition of BPP) $L \in BPP$ iff there is a deterministic polynomial-time turing-machine M and $c > 0$ s.t.*

$$x \in L \Leftrightarrow \Pr_{r\in\{0,1\}^{|x|^c}}[M(x,r) = 1] \geq \frac{2}{3}$$

Example 2.16.5. Primality testing *given $n \in \mathbb{N}$, check whether it is prime. A randomized algorithm for primality was the first famous probabilistic turing-machine . It was given by Solovay-Strassen (1970s):*

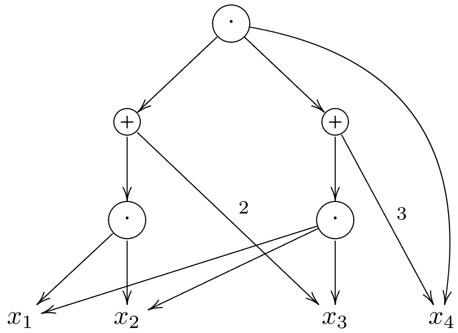
a) Pick a random $a \in (\mathbb{Z}/n\mathbb{Z})^*$

b) Output YES iff $a^{\frac{n-1}{2}} \equiv \underbrace{\frac{a}{n}}_{\textit{Jacobi-Symbol}} \pmod n$

Exercise 2.16.6. *Prove it to be a probabilistic turing-machine (none trivial)*

Today there is a deterministic polynomial-time primality test (Agrawal-Kayal-Saxena 2002).

Identity testing *Given a polynomial $C \in \mathbb{F}[x_1, \dots, x_n]$ in some "compact" form. Check whether $C = 0$. $C(x_1, \dots, x_n)$ is often presented as an arithmetic circuit: The circuit for $(x_1x_2 + 2x_3)(3x_4 + 4x_1x_2x_3)x_4$ is:*



The algorithm of Schwartz-Zippel (1980) is:

a) *Pick $(a_1, \dots, a_n) \in \mathbb{F}^n$*
b) *Output YES iff $C(a_1, \dots, a_n) = 0$*

Exercise 2.16.7. *Show that this is a probabilistic turing-machine solving identitiy testing (none trivial)*

Currently no deterministic polynomial-time algorithm is known. The corresponding language is

$$ID := \{C \mid C \textit{ is a zero arithmetic statement}\}$$

BPP captures probabilistic algorithms with two-sided error, i.e. a probabilistic turing-machine M decides $L \subseteq BPP$ then $M(x)$ could be errorneous regardess of $x \in L$ or $x \notin L$.

Definition 2.16.8. *(One-sided error, Monte-Carlo?) $L \in \mathrm{Rtime}(T(n))$ if there is a probabilistic turing-machine running in time $O(T(n))$ s.t.*

$$\begin{aligned} x \in L &\quad \Rightarrow \Pr[M \textit{ accepts } x] \geq \tfrac{2}{3} \\ x \notin L &\quad \Rightarrow \Pr[M \textit{ accepts } x] \geq 0 \end{aligned}$$

$$RP := \bigcup c \in \mathbb{N}\, \mathrm{Rtime}(n^c)$$

$$coRP := \{L \mid \overline{L} \in RP\}$$

Exercise 2.16.9. a) *$PRIMES \in coRP$*

b) *$ID \in coRP$*

c) *$RP \subseteq BPP$ and $coRP \subseteq BPP$*

d) *$RP \subseteq NP$ and $coRP \subseteq coNP$*

Definition 2.16.10. *(Zero-sided but probabilistic error, Las Vegas?) For a probabilistic turing-machine M, $\mathrm{time}_M(x)$ is a random variable (on the choices of transition steps). We say M has an expected running time T(n) if $Exp[\mathrm{time}_M(x)] \leq T(|x|)$.*

<u>$L \in \mathrm{Ztime}(T(n))$</u> *iff there is a probabilistic turing-machine M correctly deciging L in expected time $O(T(n))$.*

$$ZPP := \bigcup_{c \in \mathbb{N}} \mathrm{time}(n^c)$$

Exercise 2.16.11. *$ZPP = RP \cap coRP$*

Hint (for $\subseteq$): $L \in ZPP$, a probabilistic turing-machine M solves L in expected time $T(n)$. Run M for $2T(n)$ steps and if it has not stopped answer "YES".

Hint (for oposite): $L \in RP \cap coRP$, polynomial-time probabilistic turing-machine M_1, M_2 s.t. M_1 is always correct on $x \notin L$ and M_2 is always correct on $x \in L$. Define M' by running $M_1(x,r), M_2(x,r)$:
If $M_1(x,r) = M_2(x,r)$ then output $M_1(x,r)$
else pick another random r' and run $M'(x,r')$

Exercise 2.16.12. *$ZPP \subseteq NP \cap coNP$*

The $\frac{2}{3}$ in the definition of BPP seems arbitrary. In fact, we can fix it to anything "slightly larger" tan $\frac{1}{2}$ and get the same BPP.

Proposition 2.16.13. *(Markov's Inequality) $\Pr[X \geq k] \leq \frac{E[X]}{k}$ for any non-negative random variable X.*

<u>Proof.</u> $E[X] = \sum_{v \geq 0} v \Pr[X = v] \geq 0 + k \Pr[X \geq k]$ □

Proposition 2.16.14. *(Chernoff's bound) Let $X_1, \dots, X_k$ be independent, identically distributed boolean random-variables with $\Pr[X_i = 1] = p \forall i \in \{1, \dots, k\}$ and $\delta \in (0,1)$. Then*

$$\Pr\left[\left|\frac{\sum_{i=0}^{k} X_i}{k} - p\right| > \delta\right] < e^{-\frac{\delta^2}{4} pk}$$

<u>Proof.</u> Define $X := \sum_{i=1}^{k} X_k$. We know $E[X] = \sum_{i=1}^{k} E[X_i] = \sum_{i=1}^{k} p = kp$. Now we estimate $\Pr[X - kp > k\delta]$ and $\Pr[X - kp < -k\delta]$ for an arbitrary $\delta \in (0,1)$. Let $t > 0$ be a variable (which will be set later):

$$\begin{array}{rcl} E[e^{tX}] & := & E\left[\prod_{i=1}^{k} e^{tX_i}\right] \\ & = & \prod_{i=1}^{k} E[e^{tX_i} \\ & = & \prod_{i=1}^{k} ((1-p) \cdot 1 + pe^t) \\ & = & \prod_{i=1}^{k} (1 + p(e^t - 1)) \\ & \leq & \prod_{i=1}^{k} e^{p(e^t - 1)} \\ & = & e^{kp(e^t - 1)} \end{array}$$

As quantities are all nonnegative, we get

$$\begin{array}{rcl} \Pr[X > kp + k\delta] & = & \Pr[e^{tX} > e^{t(kp+k\delta)}] \\ & < & \frac{E[e^{tX}}{e^{t(kp+k\delta)}} \\ & < & e^{kp(e^t-1) - t(kp - k\delta)} \end{array}$$

Exercise 2.16.15. *a) Show that $t = \ln\left(1 + \frac{\delta}{p}\right)$ is minimizing the right-hand side.*

b) in a similar way estimate $\Pr[X - kp < -k\delta]$

□

Theorem 2.16.16. *Let* $L \subseteq \{0,1\}^*$ *and* M *be a polynomial-time probabilistic turing-machine s.t.* $\exists c > 0, \forall x \in \{0,1\}^*, x \in L$ *iff* $\Pr[M \textit{ accepts } x] \geq \left(\frac{1}{2} + |x|^{-c}\right)$. *Then* $\forall d > 0, \exists$ *polynomial-time turing-machine* M' *s.t.* $\forall x \in \{0,1\}^*, x \in L$ *iff* $\Pr[M' \textit{ accepts } x] \geq \left(1 - 2^{-|x|^d}\right)$

Proof. Define M': on input x run $M(x)$ for k times and let $y_1, \ldots, y_k \in \{0,1\}$ be the outputs. Now outpt for $M'(x)$ the majority of $y_1, \ldots, y_k$. Let $|x| =: n$ and we will now fix k as a function of n. Let X_i be the random variable s.t. $X_i = 1$ if y_i is correkt and $X_i = 0$ if y_i is incorrect. We know $\Pr[X_i = 1] \geq p := \frac{1}{2} + \frac{1}{n^c}$. $X_1, \ldots, X_k$ are independent and identically distributed random-variables.

By Chernoff's bound we get that

$$\begin{aligned} \Pr[M' \text{ is wrong}] &= \Pr[\text{majority is wrong}] \\ &= \Pr\left[\frac{\sum_{i=1}^k X_i}{k} < \frac{1}{2}\right] \\ &\leq \Pr\left[\left|\frac{\sum_{i=1}^k X_i}{k} - p\right| > \frac{1}{n^c}\right] \\ &< e^{-\frac{n^{-2n}}{4}\left(\frac{1}{2}+n^{-c}\right)k} \end{aligned}$$

Now it suffices to show that there is a k with $\frac{n^{-2c}}{4}\left(\frac{1}{2} + n^{-c}\right)k > n^d$. So $k > 4n^{d+2c}\left(\frac{2}{1+2n^{-c}}\right)$ must hold. So pick a k with $k > 8|x|^{d+2c}$.

□

2.17 BPP and PH

$BPP \subseteq NP$ is not known but $BPP \subseteq NP^{NP}$ is.

Exercise 2.17.1. *co*$BPP = BPP$

Theorem 2.17.2. *(Sipser-Gács 1983)* $BPP = \Sigma_2^P \cap \Pi_2^P$

Proof. We show $BPP \subseteq \Sigma_2^P$. Suppose $L \in BPP$. Then by the definition of BPP and by the error-reduction theorem we get that there is a polynomial-time turing-machine M and $m \in \mathbb{N}[X]$ s.t. $\forall x \in \{0,1\}^n$,

$$\begin{aligned} x \in L &\Rightarrow \Pr_{r \in \{0,1\}^m}[M(x,r) = 1] \geq \left(1 - \frac{1}{2^n}\right) \\ x \notin L &\Rightarrow \Pr_{r \in \{0,1\}^m}[M(x,r) = 1] \leq \frac{1}{2^n} \end{aligned}$$

Denote $S_x := \left\{r \in \{0,1\}^m \mid M(x,r) = 1\right\}$. Then the above means that $|S_x| \geq (1 - 2^{-n})2^m$ if $x \in L$ while $|S_x| \leq 2^{m-n}$ if $x \notin L$. Can we capture this in Σ_2^P?

Fix $k := \left(\frac{m}{n} + 1\right)$. For any $U = \{u_1, \ldots, u_k\} \subseteq \{0,1\}^m$ Define a Graph G_U: $V(G_U) := \{0,1\}^m$, $E(G_U) := \{(s, s') \in \{0,1\}^m \mid s \oplus u_i = s' \text{ for some } i\}$ where $\oplus$ is bitwise XOR. G_U is of degree k. For any $S \subseteq \{0,1\}^m$ define $\Gamma_U(S)$ to be the neighbours of S in G_U.

Claim 1: If $|S| \leq 2^{m-n}$ then $\forall U, |U| = k, \Gamma_U(s) \neq \{0,1\}^m$. Proof: $|\Gamma_U(S)| \leq |U||S| = k2^{m-n} < 2^m$.

Claim 2: $\exists U, |U| = k$ if $|S| \geq (1 - 2^{-n})2^m$ then $\Gamma_U(S) = \{0,1\}^m$. Proof: We prove the existence of such a $U = \{u_1, \ldots, u_k\}$ by a probabilistic argument.

Choose $U \subseteq \{0,1\}^m$ randomly. Let us fix a set S which is large enougth. Now let E_r be the event that $r \in \{0,1\}^m$ is not in $\Gamma_U(S)$ and $E_{r,i}$ the event that $r \notin S \oplus u_i$. Clearly

$$\begin{array}{rcl} \Pr_U[E_r] & = & \prod_{i=1}^k \Pr_U[E_{r,i}] \\ & = & \prod_{i=1}^k \Pr_U[u_i \notin S \oplus u_i] \\ & \leq & \prod_{i=1}^k 2^{-n} = 2^{-kn} < 2^{-m} \end{array}$$

So $\Pr_U[\exists r, r \notin \Gamma_U(S)] \leq \sum_{r \in \{0,1\}^m} \Pr_U[r \notin \Gamma_U(S)] < 1$. $\Pr_U[\forall r, r \in \Gamma_U(S)] > 0$. So there are many U s.t. $\Gamma_U(S) = \{0,1\}^*$. So Claims (1) and (2) together with 2.17 mean: $\forall x \in \{0,1\}^n, x \in L$ iff $\exists u_1, \ldots, u_k \in \{0,1\}^m, \forall r \in \{0,1\}^m, \bigvee_{i=1}^k [M(x, r \oplus u_i) = 1]$ □

Exercise 2.17.3.

- *$P \neq BPP \Rightarrow P \neq NP$*
- *$BPP = NP \Rightarrow NP = coNP \Rightarrow PH$ collapses to the first level.*

Open question: Does an BPP-complete problem exist? It is even not clear how to define such a problem.

2.18 Randomized Reductions

Definition 2.18.1. *A language A reduces to B in randomized polynomial-time, $A \leq_r B$ if there exists a polynomial-time probabilistic turing-machine M such that for all $x \in \{0,1\}^*$:*

$$\Pr[B(M(x)) = A(x)] \geq \frac{2}{3}$$

Recall the reduction of PH to $\oplus SAT$.

Definition 2.18.2. *A randomized version of NP:*

$$BP \cdot NP := \{L \mid L \leq_r SAT\}$$

Exercise 2.18.3.

- *$NP \subseteq BP \cdot NP$*
- *$coBP \cdot NP = BP \cdot coNP$*

Later we will see that $GI := Graph-Isomorphism \in NP \cap coBP \cdot NP$. This is a very natural problem which lays between P and NP and is "almost" in $NP \cap coNP$.

2.19 Randomied Space-bounded Computation

Definition 2.19.1. *A polynomial-time probabilistic turing-machine M works in space $S(n)$ if for all input $x \in \{0,1\}^*$ and random strings $r \in \{0,1\}^*$ $M(x,r)$ need work-space $\leq S(|x|)$.*

A language $L \in BPL$ if there exists a $O(\log(n))$-space probabilistic turing-machine M such that $\forall x$:

$$\Pr[M(x) = L(x)] \geq \frac{2}{3}$$

A language $\underline{L \in RL}$ *if there exists a* $O(\log(n))$*-space probabilistic turing-machine* M *such that* $\forall x$*:*

$$\begin{aligned} x \in L \Rightarrow \quad & \Pr[M(x) = 1] \geq \tfrac{2}{3} \\ x \neq L \Rightarrow \quad & \Pr[M(x) = 1] = 0 \end{aligned}$$

Exercise 2.19.2.

- $RL \subseteq NL \subseteq P$
- ** $BPL \subseteq P$ (Hint: Look at the determinant of a modified configuration graph)*

One famous RL-algorithm is for $UPath$. Let G be an indirected graph and $s, t \in V(G)$:

$$UPath := \{(G, s, t) \mid \exists \text{ path from } s \text{ to } t\}$$

Recall that $Path$ (the directed case) is NL-complete, and its $\mathbb{L}$ or RL-algorithms are open.

Theorem 2.19.3. *Aleulinas, Karp, Lipton, Lovász, Rackoff [1979]:* $UPath \in RL$

Reingold [2005]: $UPath \in \mathbb{L}$

Proof. (of $UPath \in RL$).

Suppose G is the given undirected connected graph with n vertices and $s, t \in V(G)$. The Problem is to find an undirected path from s to t in G. W.l.o.g. we can assume G to be d-regular (i.e. every vertex has d adjacent edges). For example ($d = 3$) convert a vertex of degree 5 and $\Gamma(v) = \{u_1, \ldots, u_5\}$ into three vertices v', v'', v''', connect them and then connect u_1, u_2 to v', u_3 to v'' and u_4, u_5 to v'''. Thus in logspace we can "make" G a d-regular graph ($d \geq 3$). Now the RL-algorithm for $UPath$ is:

- do a random walk starting from s, of length $n^3 \log(n)$. We will show that for all vertices $t \in V(G)$, s is connected to t $\Pr[\text{reaching } t] \geq \frac{1}{2n}$.
- we will show that

$$\begin{aligned} s \text{ is connected to } t \quad & \Rightarrow \Pr[\text{stop at } t] > \tfrac{1}{2} \\ s \text{ is not connected to } t \quad & \Rightarrow \Pr[\text{stop at } t] = 0 \end{aligned}$$

  Thus repeating this random walk, say $10n$ times, we reach t with probability $\geq \frac{2}{3}$.

Exercise 2.19.4. *Repeat the proof of error-reduction in BPP to RP for RP-algorithms where probability $\geq \frac{1}{n^c}$ can be boosted to $(1 - 2^{-n^d})$.*

We collect the probabilities $p_i := \Pr[\text{walk is at vertex } i]$ in a column vector $\vec{p} \in [0,1]^n$. At any step of the walk: $(1, \ldots, 1) \circ \vec{p} = 1$. Let $\{\vec{e^i}\}_{i=1}^n$ be the elementary vectors in $\mathbb{R}^n$ (which are zero except at the i-th position where they are one). Initially $\vec{p} = \vec{e^s}$.

W.l.o.g. we assume each vertex in G to have a self-loop.

Now let A be the "normalized adjavency matrix" of G i.e.

$$A_{i,j} := \frac{\#\text{edges between } i \text{ and } j}{d}$$

A is symmetric ($A^T = A$) with entries in $[0,1]$. and each row and column sums to 1 (such matrices are called symmetric-stochastic-matrices). If the probability vector in the current step is $\vec{p}$ then it will be $\vec{q} = A\vec{p}$ in the next step, because:

$$\begin{aligned} q_i &= \Pr[\text{walk is at } i] \\ &= \textstyle\sum_{j=0}^{n} \Pr[\text{walk is at 0 j}] \Pr[\text{walk goes to } i \mid \text{walk was at } j] \\ &= \textstyle\sum_{j=0}^{n} p_j A_{i,j} = (A\vec{p})_i \Rightarrow \vec{q} = A\vec{p} \end{aligned}$$

Then after l steps of the walk: $\vec{p} = A^l \vec{e^s}$.

We will show that $\Pr\left[\text{reaching } i \text{ in } l = 30n^2 \lg(n)\right] > \frac{1}{2n}$ for all i in the connected component of s and we will study powers of A and see how "soon" does $(A^l \vec{e^s})_t$ is a "decent" probability.

We will use the following notation

Euclidean norm $\|\vec{v}\| := \sqrt{\sum_{i=1}^{n} v_i^2}$

inner product $< \vec{u}, \vec{v} >= \sum_{i=1}^{n} u_i v_i$

Cauchy-Schwarz Inequality $\|\vec{u}\|\|\vec{v}\| \geq \sum_{i=1}^{n} |u_i v_i| \geq < \vec{u}, \vec{v} >$

Definition 2.19.5. *Let* $\vec{1} = (\frac{1}{n}, \ldots, \frac{1}{n})^T$ *be the uniform distribuation vector. Let* $\vec{1}^{\perp} := \{\vec{v} \in \mathbb{R}^n \mid < \vec{1}, \vec{v} >= 0\}$. *Define:*

$$\lambda(A) := \max\{A\vec{v} \mid \vec{v} \in \vec{1}^{\perp} \;\&\; \vec{v} \textit{ is a unit vector}\}$$

Exercise 2.19.6. *a) Show that the largest eigenvalue of* A *is* 1 *(with multiplicity* 1*)*

b) Show that the second larges eigenvalue of A *is* $\lambda(A)$

By the definition of λ:

$$\|A\vec{v}\| \leq \lambda(A)\|\vec{v}\| \quad \forall \vec{v} \in \vec{1}^{\perp}$$

Note that

$$< A\vec{v}, \vec{1} >=< \vec{v}, A\vec{1} >=< \vec{v}, \vec{1} >= 0$$

which implies that A maps $\vec{1}^{\perp}$ to itself and shrinks each vector in $\vec{1}^{\perp}$ at least by a factor of $\lambda(A) \Rightarrow \forall \vec{v} \in \vec{1}^{\perp}, \|A^l \vec{v}\| \leq \lambda(A)^l \|\vec{v}\|$

$$\Rightarrow \quad \lambda(A^l) \leq \lambda(A)^l$$

Lemma 2.19.7. *For every probability vector* $\vec{p}$*:*

$$\|A^l \vec{p} - \vec{1}\| < \lambda(A)^l$$

Remark 2.19.8. *The further* $\lambda(A)$ *is away from* 1 *the better is the "expansion property" of* G.

Proof. Decompose $\vec{p} = \alpha\vec{1} + \vec{p}'$ where $\vec{p}' \in \vec{1}^{\perp}$.

$$\begin{aligned} &\Rightarrow \quad 1 = \alpha + < \vec{p}', n\vec{1} > \Rightarrow \alpha = 1 \\ &\Rightarrow \quad A^l \vec{p} = A^l \vec{1} + A^l \vec{p}' = (\vec{1} + A^l \vec{p}') \\ &\Rightarrow \quad \|A^l \vec{p} - \vec{1}\| = \|A^l \vec{p'}\| \leq \lambda(A^l)\|\vec{p'}\| \leq \lambda(A)^l \|\vec{p'}\| \end{aligned}$$

Observe that $\|\vec{p}\|^2 = \|\vec{1}\|^2 + \|\vec{p'}\|^2$ this implies $\|\vec{p'}\| < \|\vec{p}\| < (\sum_{i=1}^n p_i)^2 = 1$ this from equation 2.19:

$$\|A^l\vec{p} - \vec{1}\| < \lambda(A)^l$$

□

What is $\lambda(A)$ in general?

Lemma 2.19.9.

$$\lambda(A) \leq (1 - \frac{1}{8dn^3})$$

Proof. Let $\vec{u} \in \vec{1}^\perp$ be a unit vector and $\vec{v'} = A\vec{v}$. We will show that

$$1 - \|\vec{v}\|^2 \geq \frac{1}{4dn^3}$$

meaning that $\|\vec{v}\| \leq \sqrt{1 - \frac{1}{4dn^3}} < 1 - \frac{1}{8dn^3}$. Which means that $\lambda(A) < 1 - \frac{1}{8dn^3}$ by definition.

$$1 - \|\vec{v}\|^2 = \|\vec{u}\|^2 - \|\vec{v}\|^2 = \sum_{1\leq i,j\leq n} A_{i,j}(u_i - v_j)^2$$

This holds because

$$\begin{array}{rcl}\sum_{i,j} A_{i,j}(u_i - v_j)^2 & = & \sum_{i,j} A_{i,j}u_i^2 - 2\sum_{i,j} A_{i,j}u_iv_j + \sum_{i,j} A_{i,j}v_j^2 \\ & = & \|\vec{u}\|^2 - 2 < A\vec{u}, \vec{v} > + \|\vec{v}\|^2 \\ & = & \|\vec{u}\|^2 - 2\|\vec{v}\|^2 + \|\vec{v}\|^2 \\ & = & \|\vec{u}\|^2 - \|\vec{v}\|^2\end{array}$$

Thus it suffices to show some i, j s.t. $A_{i,j}(u_i - v_j)^2 \geq \frac{1}{4dn^3}$. Because G has self-loops, $A_{i,i} = \frac{1}{d}$ and we can further assume $|u_i - v_i| < \frac{1}{2n^{1.5}}$ for all i, otherwise we are done.

Sort the coordinates of $\vec{u}$ w.l.o.g. $u_1 \geq u_2 \geq \ldots \geq u_n$. Since $\sum_{i=1}^n u_i = 0$ and $\sum_{i=1}^n u_i^2 = 1$ $u_1 > 0$, $u_n < 0$ and either $u_1 \geq \frac{1}{\sqrt{n}}$ or $u_n \leq -\frac{1}{\sqrt{n}}$ so $u_1 - u_n \geq \frac{1}{\sqrt{n}}$.

$$\begin{array}{ll}\Rightarrow & \exists i_0 : u_{i_0} - u_{i_0+1} \geq \frac{1}{n^{1.5}} \\ \Rightarrow & \forall i \in \{1, \ldots, i_0\} \ \& \ \forall j \in \{i_0 + 1, \ldots, n\}, u_i - u_j \geq \frac{1}{n^{1.5}}\end{array}$$

□ Since G is connected there exists and edge $(i, j) \in \{1, \ldots, i_0\} \times \{i_0 + 1, \ldots, n\}$ with $A_{i,j} = \frac{1}{d}$ & $u_i - u_j \geq \frac{1}{n^{1.5}}$.

Note that $|u_i - v_j| \geq |u_i - u_j| - |u_j - v_j| > \frac{1}{n^{1.5}} - \frac{1}{2n^{1.5}} = \frac{1}{2n^{1.50}}$. Which finally shows that $A_{i,j}(u_i - v_j)^2 \geq \frac{1}{d}\frac{1}{4n^3}$ and $1 - \|\vec{v}\|^2 > \frac{1}{4dn^3}$. □

Lemma 2.19.10. *Do a random walk on G from s for $l \geq 10dn^3 \lg(n)$ steps. Then*

$$\Pr[\textit{reaching } t \textit{ at the } l\textit{-th step}] > \frac{1}{2n}$$

Proof. Let $\vec{p}$ be the probability distribution on $V(G)$ at the l-th step. By 2.19.7 and 2.19.9:

$$\|A^l\vec{e^s} - \vec{1}\| < \left(1 - \frac{1}{8dn^3}\right)^l < \frac{1}{2n^{1.5}}$$

and by Cauchy-Schwartz

$$\begin{array}{rrcl} & \sum_{i=1}^n |(A^l\vec{e^s} - \vec{1})_i| & < & \frac{1}{2n} \\ \Rightarrow & |(A^l\vec{e^s} - \vec{1})_t| & < & \frac{1}{2n} \\ \Rightarrow & (A^l\vec{e^s})_t & > & \frac{1}{n} - \frac{1}{2n} = \frac{1}{2n}\end{array}$$

□

Remark 2.19.11. *a) So we have seen that $UPath \in RL$ [1979]*

b) Now, $UPath \in \mathbb{L}$ is known
Idea: If we can transform G to a graph G' (in $\mathbb{L}$) with a constant $\lambda(G')$ (again this is the second largest eigenvalue of the normalized adjacency-matrix) then it will have a much better "expansion", i.e.

$$\left|A^l \vec{e^s} - \vec{1}\right| < \lambda(G')^l \le \frac{1}{n^2}$$

for $l \in O(\lg(n)), n := |V(G')|$. Prooved by Reingold [2005]. The transformation will be shown in the next course.

c) An expander is a family of graphs $\{G_i\}_{i=1}^{\infty}$ with $\lambda(G_i) < \epsilon \in \mathbb{R}\ \forall i \in \mathbb{N}$

2.20 Graph Isomorphism (GI)

Definition 2.20.1.

$$\begin{aligned} GI &:= \{(G_1, G_2) | \textit{finite graphs } G_1 \cong G_2\} \\ GNI &:= \{(G_1, G_2) | \textit{finite graphs } G_1 \ncong G_2\} \end{aligned}$$

It is trivial that $GI \in NP$ and $GNI \in coNP$ but it is open if $GNI \in NP$.

Theorem 2.20.2. *(Goldwasser-Sipser [1986])*

$$GNI \in BP \cdot NP$$

Proof. We will show the existence of a polynomial-time transformation A and a constant $c > 0$ s.t.

$$\begin{aligned} (G_1, G_2) \in GNI &\Rightarrow \Pr_r[\exists y, A(G_1, G_2, r, y) = 1] \ge \tfrac{2}{3} \\ (G_1, G_2) \notin GNI &\Rightarrow \Pr_r[\exists y, A(G_1, G_2, r, y) = 1] \le \tfrac{1}{3} \end{aligned}$$

where $r, y \in \{0,1\}^{|(G_1, G_2)|^c}$.

The key idea is to look at a set S associated with the graphs G_1 and G_2 (having n vertices):

$$\begin{aligned} S := \{(H, \pi) \mid & H \text{ is a graph on vertices } \{1, \ldots, n\} \\ & H \cong G_1 \vee H \cong G_2, \pi \in \mathrm{Aut}(H)\} \end{aligned}$$

- If $G_1 \cong G_2$ then $\#S = \#(\{H | H \cong G_1\} \times \mathrm{Aut}(G_1)) \Rightarrow \#S = \frac{n!}{\#\mathrm{Aut}(G_1)} \cdot \#\mathrm{Aut}(G_1) = n!$
- If $G_1 \ncong G_2$ then $\#S = \sum_{i=1}^{2} \#(\{H | H \cong G_i\} \times \mathrm{Aut}(G_i)) = 2n!$.
- And "membership-in-S" $\in NP$.

Thus $\#S$ is larger (by a factor of 2) when $(G_1, G_2) \in GNI$. We now give a general method for set-lower-bound in $BP \cdot NP$.

Recall from the proof of $SAT \le_r \oplus SAT$, a hash function for $B \in \{0,1\}^{k \times m}, b \in \{0,1\}^k$ (also see 2.15.15):

$$\begin{aligned} h_{B,b}: \{0,1\}^m &\to \{0,1\}^k \\ u &\mapsto (Bu + b) \end{aligned}$$

with the above properties.

Let $m = \max_{s \in S} \{|s|\}$ (where we think of S as a subset of $\{0,1\}^m$). And let $k \in \mathbb{N}$ be such that $2^{k-2} \leq 2n! \leq 2^{k-1}$.

Idea: For a random hash-function $h_{B,b}$ and a random image z there exists a pre-image with high-probability iff S is "large".

- If $\#S = 2n!$ then
$$\Pr_{B,b,z}\left[\exists y \in S : h_{B,b}(y) = z\right] \geq \frac{\#S}{2^k}\left(1 - \frac{\#S}{2^{k+1}}\right) \geq \frac{3}{16}$$
.
- If $\#S = n!$ then
$$\Pr_{B,b,z}\left[\exists y \in S : h_{B,b}(y) = z\right] \leq \frac{\#S}{2^k} = \frac{n!}{2^k} \leq \frac{1}{4}$$
- It is easy to see that repeatuing this, say 8 times gives us:
$$\begin{array}{rlcc} \#S = 2n! \Rightarrow & \Pr_{\vec{B},\vec{b},\vec{z}}\left[\exists \vec{y} \in S^8 : h_i(y_i) = u_i \ \forall i\right] & > & \frac{2}{3} \\ \#S = n! \Rightarrow & \Pr_{\vec{B},\vec{b},\vec{z}}\left[\exists \vec{y} \in S^8 : h_i(y_i) = u_i \ \forall i\right] & < & \frac{1}{3} \end{array}$$
- We can check $h(y) = z$ in polynomial-time and $y \in S$ too, using a certicate. Thus we get a suitable polynomial-time turing-machine A.

□

So we see that $GNI \in coNP \cap BP \cdot NP$

Theorem 2.20.3. *(Schöning [1987]) If GI is NP-complete then $\Sigma_2^P = \Pi_2^P$.*

Proof. Suppost GI is NP-complete then GNI is $coNP$-complete. Let $\psi := \exists x \forall y \phi(x,y)$ (where ϕ is a boolean formula) be a Σ_2^P-instance with $n := |x| = |y|$.

We can convert the question of $\forall y \phi(x,y)$ to an equivalent question of graphs g: $g(x) \in GNI$.

Now define a new quantifier M by $M(z \in \{0,1\}^m) : \tau(z)$ is true iff $\tau(z)$ is true for "most" $z \in \{0,1\}^m$ (probability bound will depend on the context).

Since $GNI \in BP \cdot NP$ (M, $\exists$) we can reqrite $g(x) \in GNI$ as
$$Mr\exists a : T(x,r,a) = 1$$
for some polynomial-time turing-machine T.

Thus $\psi_1 := \exists x Mr \exists a T(x,r,a) = 1$ is equivalent to ψ. By a suitable error reduction, we get a polynomial-time turing-machine T' and bigger strings r', a' s.t. $\psi_2 := Mr' \exists x \exists a' T'(x,r',a') = 1$ is equivalent to ψ_1.

Exercise 2.20.4. *Work out the amplification under which ψ_1 and ψ_2 become equivalent.*

Recall the proof of 2.17.2), it can be seen as a general way to replace M by $\forall\exists$: $\psi_3 := \forall s_1 \exists s_2 \exists s_3 \exists s_4 T''(\vec{s}) = 1$ is equivalent to ψ_2, hence to ψ.
$$\begin{array}{rrcl} \Rightarrow & \Sigma_2^P SAT & \in & \Pi_2^P \\ \Rightarrow & \Sigma_2^P & \subseteq & \Pi_2^P \\ \Rightarrow & \Sigma_2^P & = & \Pi_2^P \end{array}$$
$\Rightarrow PH$ collapses. □

Chapter 3

Circuits

3.1 Definition of Boolean & Arithmetic Circuits

Turing machines capture problems that can be solved by some algorithm. What about a language L for which there is a different algorithm A_n for each $x \in \{0,1\}^*$ with $n = |x|$ (non-uniform vs. uniform computation). There is a mathematically elegant way to capture these problem: circuits (inspired by real "‘silicon chips"’).

Definition 3.1.1. *A boolean circuit $C(x_1, \ldots, x_n)$ is a directed rooted tree with boolean input nodes $x_1, \ldots, x_n$ at the leaves. The inernal nodes are labeled with OR, AND and NOT and the ouput is calculated at the root. Sometimes a boolean circuit can have several outputs and so it has several roots.*

The maximal indegree of a node is the fanin of C and the maximal outdegree the fanout. $|C| := \mathrm{size}(C) := \#$*nodes in the tree.* $\mathrm{depth}(C) := \#$*levels in the tree. A circuit is called a formula if its fanout is 1. A boolean circuit with n inputs and m outputs computes a function $\{0,1\}^n \to \{0,1\}^m$.*

Now we can formalize when a circuit family solves a problem:

Definition 3.1.2. *Let $S : \mathbb{N} \to \mathbb{N}$ be a function. A $s(n)$-sized circuit family is a sequenz of boolean circuits $\{C_n\}_{n\in\mathbb{N}}$ s.t. $|C_n| = O(s(n))$.*

$$\begin{aligned}\mathrm{size}(s(n)) := \quad & \{L \subseteq \{0,1\}^* \mid \exists s(n) - \textit{sized circut family } \{C_n\}_{n\in\mathbb{N}} \\ & \textit{s.t. } \forall x : C_n(x) = L(x) \textit{ where } n = |x|\}\end{aligned}$$

$$P/poly := \bigcup_{c\in\mathbb{N}} Size(n^c)$$

non-uniform polynomial-time.

Exercise 3.1.3. *a) Any language L has a $n \cdot 2^n$-sized circuti family i.e.* $\mathrm{size}(n2^n) = 2^{\{0,1\}^*}$.

b) Uncomputable problems are solvable with $n2^n$-circuit families

c) there exists an uncomputable problem solvable by an n-sized circuit family

Hint: Consider

$$L := \{1^n \mid M_n(1^n) \textit{ where } M_n \textit{ is a turing-machine described by } (n)_2\}$$

So constant-sized circuits can solve some uncomputable problems.

We reduce the strength of $P/poly$ by a uniformity restriction.

Definition 3.1.4. *A circuit family $\{C_n\}$ is called logspace-uniform if there exists a logspace algorithm that generates C_n on input 1^n.*

$$(\textit{logspace-uniform})P/poly := \{L \subseteq \{0,1\}^* \mid \exists c > 0, \exists \text{ logspace-uniform } n^c\text{-sized circuit family } \{C_n\} \text{ s.t. } C_n \text{ decides } L \cap \{0,1\}^n\}$$

Theorem 3.1.5. $(\textit{logspace-uniform})P/poly = P$

Proof. "'$\subseteq$"' Even with polynomial-time in place of logspace it is true!

"'$\supseteq$"' Let $L \in P$ be a language and M is a n^c-time turing-machine deciding L. Idea: We encode the steps of M on $\{0,1\}^n$ in a polynomial-sized circuit C_n in $O(\log(n))$ space.As in Cook-Levins (proof of NP-hardness of SAT) first convert M into an n^{2c}-time turing-machine $\tilde{M}$ which is oblivious (i.e. the i-th head-movement of $\tilde{M}$ depends only on i). Thus the i-th head-position of $\tilde{M}$ can be computed in $O(\log(n))$ space given 1^i. Let $\zeta_1, \dots, \zeta_{n^{2c}}$ be the n^{2c} configurations of $\tilde{M}$ starting from the initial to the final state of $\tilde{M}$ on input $x_1, \dots, x_n$. Each ζ_i is an array (head-position, head-bit, state). Construct the circuit C_n as: In the i-th level it computes ζ_i from ζ_{i-1} using $\delta(\tilde{M})$. Finally it outputs 1 iff $\zeta_{n^{2c}}$ is in the accept state. So $|C_n| = \text{depth}(C_n) = O(n^{2c})$. C_n can be generated given 1^n in $O(\log(n))$ space. And we see that L has a logspace non-uniform n^c-sized circuit family. □

Exercise 3.1.6. *a) $P \subseteq P/poly$*

b) $SAT \notin P/poly \Rightarrow P \neq NP$

Open: $SAT \notin P/poly$

Remark 3.1.7. *SAT can have a polynomial-sized circuit even if $SAT \notin P$.*

Theorem 3.1.8. *(Karp-Lipton, 1980) $NP \subseteq P/Poly \Rightarrow \Pi_2 = \Sigma_2 \Rightarrow PH = \Sigma_2$*

Proof. Suppose $SAT \in P/Poly$ then for any boolean formula $\phi(x_1, \dots, x_n)$ of size m there is a constant c and a boolean circuit $C_m(\phi(x_1, \dots, x_n))$ of size m^c s.t. $C_m(\phi) = 1$ iff ϕ is satisfiable.

Note that $\phi(x_1, \dots, x_n)$ is satisfiable iff $\phi(1, x_2, \dots, x_n)$ or $\phi(0, x_2, \dots, x_n)$ is satisfiable (or both). So using SAT as an oracle you can actually find a satisfying assignment. This property is called self-reducibility.

By repeating this n times we get a circuit C'_m of size $O(nm^c)$ that outputs a satisfying assignment of ϕ (if one exists). C'_m can be expressed using m^{3c} bits.

Let $\forall u \exists v \psi(u, v)$ be a $\Pi_2 SAT$ instance. Look at the formula $\forall[\psi(u, C'_m(\psi)) = 1]$. Now look at $\exists w \forall u[w$ is a circuit of size $\leq m^{3c}$ & $\psi(u, w(\psi)) = 1]$. The two quantified formulars are equivalent. $\Pi_2 \subseteq \Sigma_2 \Rightarrow PH = \Sigma_2$. □

This theorem gives hope that "SAT does not have polynomial-sized circuits". There are results known for SAT not having certein special booolean circuits, e.g. monotone circuits.

It is interesting that most of the functions $\{0,1\}^n \to \{0,1\}$ have high circuit complexity.

Theorem 3.1.9. *"Almost all" boolean functions* $\{0,1\}^n \to \{0,1\}$ *require circuits of size* $\geq 2^n/10n$.

Proof. $\#\left\{f : \{0,1\}^n \to \{0,1\}\right\} = 2^{2^n}$ and w.l.o.g. fanin of every circuit is 2 and fanout is 1 (by breaking a circuit with more fanout / fanin up in more levels) then $\#\{\text{circuits of size } s \text{ on } n-\text{sized input}\} < s^{3s}$. For $s = \frac{2^n}{10n}$, $s^{3s} \leq 2^{\frac{2^n}{3}} < 2^{2^n}$. Thus a "random" function requires circuits of size $> \frac{2^n}{10n}$. □

Open: Find such a function explicitly.

Exercise 3.1.10. *Proove the non-uniform hierarchy theorem:*
$\forall T, T' : \mathbb{N} \to \mathbb{N}$ *s.t.* $T'(n) = \omega(T(n)\lg(T(n)))$ *then* $\text{size}(T(n)) \subsetneq \text{size}(T'(n))$
Hint: By counting a function in $\text{size}(T'(n)) - \text{size}(T(n))$

BPP is also related to $P/poly$:

Theorem 3.1.11. *(Adleman 1918)* $BPP \subseteq P/poly$

Proof. Suppose $L \in BPP$ then there exists a polynomial-time turing-machine M s.t. $\forall n \in N, x \in \{0,1\}^n$:

$$\begin{array}{lll} x \in L & \Rightarrow & \Pr_{r:|r|=m}[M(x,r) = 1] \geq \left(1 - \frac{1}{2^{n+1}}\right) \\ x \notin L & \Rightarrow & \Pr_{r:|r|=m}[M(x,r) = 1] \leq \frac{1}{2^{n+1}} \end{array}$$

Say $r \in \{0,1\}^m$ is bad if for some $x \in \{0,1\}^n$: $M(x,r)$ is wrong. Thus

$$\#\{r \mid r \text{ is bad}\} \leq 2^n \frac{2^m}{2^{n+1}} = \frac{2^m}{2}$$

So pick an $r_n \in \{0,1\}^m$ for which $M(x, r_n)$ is correct for all $x \in \{0,1\}^n$. Define C_n to be the circuit that simulates $M(x_1, \ldots, x_n, r_n)$ (where x_i is the i-th bit of x). Which implies that $LinP/poly$. □