Complexity

doc. RNDr. Ondřej Čepek, Ph.D.

June 29, 2021

Contents

1 Repetition, NS hierarchy

1.1 Abbreviations

- TM - turing machine.
- DTM - deterministic turing machine.
- NTM - non-deterministic turing machine.
- DS - DSPACE
- NS - NSPACE
- PS - deterministic polynomial space
- DT - DTIME
- NT - NTIME
- QBF - quantifiable boolean formula
- CNF - conjunctive normal form
- PH - polynomial hierarchy
- PTM - Probabilistic TM
- r.v. - Random variable
- RP - Randomized Polynomial time
- ZPP - Zero-error Probabilistic Polynomial time

Theorem 1.1 (NT and DS relation).

$$\forall f : \mathbb{N} \to \mathbb{N} : NT(f(n)) \subseteq DS(f(n))$$

Proof. Construct TM with following algorithm:

1. k = 1
2. do
3. foreach y branch $|y| = k$
4. Simulate M(x) by y
5. If(Acc) accept
6. k++;
7. until all simulations of M(x) by y that rejected // if all branches rejected $\Rightarrow$ there is no accepting computation
8. reject

M works in time $g(n) = \mathcal{O}(f(n))$, computation in step 4. will do at most g(n) steps.
Space:
1) to store k we need $g(n)$ of space, $k \leq g(n)$.
2) To simulate $M(x)$ by y we need $\mathcal{O}(g(n))$ space.
Correctness:
We want constructed DTM to accept the same language as initial NTM. If there is an accepting branch of NTM, DTM also accepts by some branch y. Otherwise we reach $k = g(n)$ and DTM rejects.
TM cannot loop forever by the definition of time complexity (we restrict the languages to recursive).
Technical details:
Constructed TM should have 2 tapes:
a) working tape
b) tape to store y (vector that encode branch of NTM) □

Note 1.2. If we are constrained to use single tape, any k tape machine can be compressed to 1 tape machine. We need 2 steps:

1. compress $\Sigma \to \Sigma^k$
2. How to deal with many heads of the original TM? $\Rightarrow$ simulate 1 step of the original machine by many steps in new TM. In total, time complexity is $\mathcal{O}(n^2)$ and space complexity is $\mathcal{O}(n)$.

Theorem 1.3 (NS and DT relation).

$$\forall f : \mathbb{N} \to \mathbb{N}, \forall L \in NS(f(n)) \Rightarrow \exists c_L : L \in DT\left(c_L^{f(n)}\right)$$

Proof. All accepting configuration leaves are bounded by the # of configurations. However, # of all paths $2^{c_L^{f(n)}}$. How to avoid double exponent? $\Rightarrow$ Perform BFS in configuration graph. # of edges could be quadratic , however

$$\left(c_L^{f(n)}\right)^2 = \left(c_L^2\right)^{f(n)}$$

c_L^2 is another constant. □

Definition 1.4 (Universal TM). Input: (x, y), where x is Godel number of TM to be simulated. y is input of TM.
Alphabet: $\Sigma = \{0, 1\}$.
Working tapes: 3

1. Tape with transition table of M_x.
2. Tape with current state q
3. Working tape

If $S(n)$ is space used by $M_x \Rightarrow$ we need $\max(\lceil \log(t) \rceil, S(n), |x|)$.
TODO what is t?

Observation 1.5. If initial TM has k tapes and time complexity is $T(n)$ we can compress to 2 tapes with a cost of time complexity being $\mathcal{O}(T(n)\log(n))$.
// TODO write proof (moving blocks on tape)
Therefore simulating TM with multiple tapes could be reduces to 2 tapes, then simulate on Universal TM. Time complexity of Universal TM is dominated by finding the transition $\Rightarrow \mathcal{O}(|x| \cdot T(n))$.

Definition 1.6 (Space constructible function). Function f is *Space constructible* $\iff \exists$ TM with unary alphabet which marks exactly $|f(n)|$ cells on the working tape. E.g. $\log, e^x$, polynomials.

Theorem 1.7 (Space hierarchy). *Let S_1, S_2 are space constructible functions and*

$$S_1 \in o(S_2) \Rightarrow DS(S_1(n)) \subsetneq DS(S_2(n))$$

Proof. Construction by Cantor diagonalization method $\Rightarrow$ find a language that is different from all in $\mathcal{O}(S_1)$. □

Definition 1.8 (Time constructible function). Function f is *Time constructible* $\iff \exists$ TM that does exactly $|f(n)|$ steps. You can think of it as alarm clock.

Observation 1.9. Every time constructible function is space constructible.

Theorem 1.10 (Time hierarchy). *Let S_1, S_2 are time constructible functions and*

$$T_1 \cdot \log(T_1(n)) \in o(T_2) \Rightarrow DT(T_1(n)) \subsetneq DT(T_2(n))$$

Note that $\log(T_1(n))$ is required because of k to 2 tapes compression.

Proof. Construction by Cantor diagonalization method $\Rightarrow$ find a language that is different from all in $\mathcal{O}(T_1)$. □

Theorem 1.11 (Savic). *Under some mild assumptions (space constructibility, functions bigger than $\log(n)$) following statement is true:*

$$NS(f(n)) \subseteq DS(f^2(n))$$

Proof. Find a path in configuration path. We cannot use neither BFS not DFS as the complexity is linear in edges which is exponential comparing to input. Therefore we use recursive algorithm which for all states K reachable by a path of length

$$\frac{c_L^{f(n)}}{2^i}$$

tries to find a path from $C_{init} \to K \to C_{accept}$.
As we divide path by 2 at every recursive call, recursion tree height is equal to $\log_2(n)$. Therefore time complexity is

$$\log_2(c_L^{f(n)}) = f(n) \cdot \log_2(c_L)$$

On each level of recursion we have to store C_{init}, K, C_{accept} which requires $\mathcal{O}(f(n))$ space. Having $\mathcal{O}(f(n))$ levels, total space complexity is $\mathcal{O}(f^2(n))$. □

Note 1.12. Time version of Savic would imply $P = NP$.

1.2 NS hierarchy

Lemma 1.13 (Translation lemma). *Let $S_1(n), S_2(n), f(n)$ be space constructible functions, also*

$$S_2(n) \geq n, f(n) \geq n$$

Then

$$NS(S_1(n)) \subseteq NS(S_2(n)) \Rightarrow NS(S_1(f(n))) \subseteq NS(S_2(f(n)))$$

Lemma allows to replace $n \to f(n)$.
We can also prove Translation Lemma for DS, NT, DT.

Proof. Let $L_1 \in NS(S_1(f(n)))$ arbitrary, L_1 is recognized by NTS M_1 in space $S_1(f(n))$. We want to prove that $L_1 \in NS(S_2(f(n)))$ by constructing $L_2 \in NS(S_1(n))$ using padding. Define $L_2 := \{x\triangle^i | M_1 \text{ accepts } x \text{ in space } S_1(|x|+i)\}$. Where $\triangle$ is a new symbol, that $\notin$ initial Σ.
Algorithm of $M_2, L(M_2) = L_2$ on input $x\triangle^i$ is the following:

1. Mark $S_1(|x|+i)$ cells on tape.
2. Simulate M_1.
3. if($M_1(x)$ accept $\wedge$ did not use more space than marked in 1) then accept

From the construction, M_2 recognizes L_2 using $S_1(n)$ space. Consequently, $L_2 \in NS(S_1(n))$. Combining with assumption of the lemma

$$L_2 \in NS(S_2(n))$$

And there exists a TM M_3 that recognizes L_2 using $S_2(n)$ space.
The last step is to construct M_4 that recognizes L_1 using $S_2(f(n))$ space. M_4 has 2 working tapes and has following algorithm (M_4 on input x):

1. Mark $f(n)$ cells on 1st tape.
2. Mark $S_2(f(n))$ cells on 2nd tape. Using 1st tape as "input".
3. foreach $x\triangle^i, i = 0, 1, \ldots$
   simulate M_3 on input $x\triangle^i$. If head of M_3 is inside x M_4s head is in the same place. Otherwise, head of M_3 is inside padding, so M_4 uses counters to track M_3s position. Counter has length at most $1 + \log i$.
4. if($M_3(x)$ accept) then M_4 accept.
5. else if($1 + \log(i) \leq S_2(f(n))$) then $++i$;
   Counter did not overflow and is less than $S_2(f(n))$.
6. else if($1 + \log(i) > S_2(f(n))$) then reject;

If M_4 accepted input x, M_3 accepted $x\triangle^i$ for some $i \Rightarrow x\triangle^i \in L_2 \Rightarrow M_1$ accepts $x \Rightarrow x \in L_1$.
On the other hand, if $x \in L_1 \Rightarrow x\triangle^i \in L_2$ for $i = f(|x|) - |x|$ therefore counter i requires

$$\log(f(|x|) - |x|) \leq S_2(f(|x|))$$

space.

□

Theorem 1.14 (NS hierarchy for polynomials). *Let $\varepsilon > 0, r > 1$. Then*

$$NS(n^r) \subsetneq NS(n^{r+\varepsilon})$$

Proof. From the density of rationals:

$$\exists s,t \in \mathbb{N} : r \leq \frac{s}{t} \leq \frac{s+1}{t} \leq r+\varepsilon$$

It is sufficient to prove that:

$$NS(n^{\frac{s}{t}}) \subsetneq NS(n^{\frac{s+1}{t}})$$

Assume by contradiction

$$NS(n^{\frac{s+1}{t}}) \subseteq NS(n^{\frac{s}{t}})$$

Now we use lemma 1.13 for

$$S_1 = n^{\frac{s+1}{t}}, S_2 = n^{\frac{s}{t}}, f(n) = n^{(s+i)t}, i = 0, 1 \ldots, t$$

Note 1.15.

Note 1.16.

We get

$$\forall i : NS((n^{(s+i)t})^{(s+1)/t}) \subseteq NS((n^{(s+i)t})^{s/t}) \Rightarrow \forall i : NS(n^{(s+i)(s+1)}) \subseteq NS(n^{(s+i)s})$$

Now we write for all i:

- $i = 0 : NS(n^{s(s+1)}) \subseteq NS(n^{s^2})$
- $i = 1 : NS(n^{(s+1)(s+1)}) \subseteq NS(n^{(s+1)s})$
- $\ldots$
- $i = s : NS(n^{2s(s+1)}) \subseteq NS(n^{(2s)s})$

From the exponents we conclude that for every inequality right side is a subset of left side. So we get a chain of subsets and can use Savic theorem to get a contradiction:

$$NS(n^{(s+1)2s}) \subseteq NS(n^{s^2}) \overset{Savic}{\subseteq} DS(n^{2s \cdot s}) \overset{spaceH}{\subsetneq} DS(n^{2s \cdot s + 2s}) \subseteq NS(n^{2s(s+1)})$$

As the beginning and the end are the same sets and the chain of subsets has strict inclusion.

□

2 Relative computations by Oracle TM

Definition 2.1 (Oracle TM). Oracle TM is a DTM with an Oracle A (where A is a language) differs from an ordinary DTM by the following:

- Oracle tape (with same alphabet as TM)
- 3 special states: QUERY, YES, NO
- In QUERY state TM moves to YES state if word on the oracle tape $\in A$ (moves to NO o/w). After the answer oracle tape is erased (to reuse space in Space complexity).
- Language of the accepted word by an oracle TM M is $L(M, A)$.

Note 2.2. For NTM definition works the same.

Note 2.3. Ordinary DTM is the same as oracle DTM with $A = \emptyset$.

Consider now a comparison of the oracle DTM, when oracle language A is *not fixed in advance*. Computation forms a tree, that branches at every QUERY.

Observation 2.4. Consider NTM vs Oracle DTM.
"$\Rightarrow$". If NTM M has language $L(M)$, set oracle language $A = L(M)$. "$\Leftarrow$". If oracle language is not recognizable (e.g. HALT), we cannot simulate such NTM.

Definition 2.5 (Turing reducibility). *Turing reducibility* - let A,B languages. We say that A is (deterministically) Turing reducible to B in polynomial time if there $\exists$ an oracle DTM M working in polynomial time such that

$$A = L(M, B), A \leq^T B$$

Example 2.6. $A \in P \Rightarrow A \leq^T \emptyset$. Since we have polynomial time algorithm without any oracle.

Definition 2.7 ($\mathbb{P}(A)$). Let A be a language, then

$$\mathbb{P}(A) = \{B | B \leq^T A\}$$

Definition 2.8 ($\mathbb{P}(\mathcal{C})$). Let $\mathcal{C}$ be a set of languages then

$$\mathbb{P}(\mathcal{C}) = \{B | \exists A \in \mathcal{C} : B \leq^T A\}$$

Theorem 2.9 ($\mathbb{P}(\mathbb{P}) = \mathbb{P}$).

$$\mathbb{P}(\mathbb{P}) = \mathbb{P}$$

Proof. $\mathbb{P} \subseteq \mathbb{P}(\mathbb{P})$. Let $A \in \mathbb{P}$, use A as an oracle with 1 QUERY or use empty oracle.
$\mathbb{P}(\mathbb{P}) \subseteq \mathbb{P}$. Let $B \in \mathbb{P}(\mathbb{P}) \iff \exists A \in \mathbb{P} \exists ODTM\ M : B = L(M, A)$.
To prove the inclusion, we have to construct ordinary DTM that recognizes B. Such TM M_B simulates M and whenever M enters QUERY state, simulate M_A to check if word w on oracle tape

$$w \in L(M_A) = A$$

Now we have to check time complexity.
M makes $p(|x|)$ queries (for p polynomial), as the total number of steps is polynomial. Each query word length is at most $p_w(|x|) = t$. Every query has at most $p_q(|t|)$ steps. In total, query is $p_q(p_w(|x|))$. And the total time complexity of the TM is $p(p_q(p_w(|x|)))$. Which is also polynomial. □

Definition 2.10 (Turing reducibility (N)). *Turing reducibility (non-deterministic)* - let A,B languages. We say that A is non-deterministically Turing reducible to B in polynomial time if there $\exists$ an oracle NTM M working in polynomial time such that

$$A = L(M, B), A \leq^{NP} B$$

Definition 2.11 ($\mathbb{NP}(A)$). Let A be a language, then

$$\mathbb{NP}(A) = \{B | B \leq^{NP} A\}$$

Definition 2.12 ($\mathbb{NP}(\mathcal{C})$). Let $\mathcal{C}$ be a set of languages then

$$\mathbb{NP}(\mathcal{C}) = \{B | \exists A \in \mathcal{C} : B \leq^{NP} A\}$$

Note 2.13. Relativized definition also works for other classes, e.g. EXPTIME.

Definition 2.14 (PS).

$$PS = \bigcup_{i=0}^{\infty} DS(n^i) = NPS = \bigcup_{i=0}^{\infty} NS(n^i)$$

Where the 2nd equality holds because of Savic theorem 1.11.

Definition 2.15 (PS(A)). $PS(A) = \{B | B$ accepted by an oracle DTM working in polynomial space, st $B = L(M, A)\}$.
Also for class of languages $\mathcal{C}$.

Note 2.16.

$$\mathbb{P} \subseteq \mathbb{NP} \subseteq PS$$

Where last inclusion hold because of $NT(f(n)) \subseteq DS(f(n))$.
Same proof but as for ordinary TM, but with oracle TM that shares same oracle language A.

Observation 2.17 (Open question). What about $\mathbb{NP}(\mathbb{NP})$? Still an open question, depends on $\mathbb{P} = \mathbb{NP}$.
We cannot simply plug NTM back to the original TM with oracle, as NTM serving as an oracle could have multiple accepting or rejecting leaves.

3 Polynomial Hierarchy

3.1 Simplified PH

Definition 3.1 (Simplified PH). Consider a sequence

$$\Sigma_0^P, \Sigma_1^P, \Sigma_2^P, \dots$$

Where $\Sigma_0^P = \mathbb{P}, \Sigma_{i+1}^P = \mathbb{NP}(\Sigma_i^P)$.
And Polynomial hierarchy(simplified) is:

$$PH = \bigcup_{i \geq 1} \Sigma_i^P$$

Theorem 3.2 (Polynomial hierarchy(simplified)).

$$PH \subseteq PS$$

In plain words, PH is only smth in between NP and PS.

Proof. By induction prove $\forall i : \Sigma_i \subseteq PS$.

- $i = 0 \Rightarrow \mathbb{P} \subseteq PS$
- $i \to i+1$, assume $\Sigma_i \subseteq PS$.

By definition: $\Sigma_{i+1} = \mathbb{NP}(\Sigma_i)$ Then

$$\mathbb{NP}(\Sigma_i) \subseteq \mathbb{NP}(PS)$$

We made set of oracles larger, set of recognized languages cannot shrink.
Now we use $\forall C : \mathbb{NP}(C) \subseteq PS(C)$.
Therefore

$$\mathbb{NP}(C) \subseteq PS(PS) \subseteq PS$$

Last inclusion is up to prove (similar to $\mathbb{P} = \mathbb{P}(\mathbb{P})$):

$$B \in PS(PS) \iff \exists A \in PS \exists DTMM : B = L(M, A)$$

Where M works in poly space.

$$A \in PS \iff \exists DTM\ M_A : A = L(M_A)$$

Where M_A works in poly space. Same in $\mathbb{P} = \mathbb{P}(\mathbb{P})$ proof 2.9 replace oracle QUERY by DTM computation (which accepts or rejects)
The last thing is to check that used space is polynomial. Which is true since $\forall t \in QUERY$: $|t| \leq p(|x|)$ for some polynomial p. Space taken by DTM M_A that computes the QUERY is $p_t(|t|) \leq p_t(p(|x|))$. Also, we can ask exponentially many QUERies, however the space is reused, therefore space is bounded by the largest QUERY. □

3.2 Time and Space classes relation

Reminder 3.3 (Space classes).

$$\begin{aligned} LOG &= DS(\log n) \\ NLOG &= NS(\log n) \\ POLYLOG &= \bigcup_{i \geq 0} DS(\log^i n) \\ PS &= \bigcup_{i \geq 0} DS(n^i) \\ NSSP &= \bigcup_{i \geq 0} NS(n^i) \\ EXPSPACE &= \bigcup_{i \geq 0} DS(2^{ni}) \end{aligned} \tag{1}$$

Reminder 3.4 (Time classes). No way to define LOG class, as we have to read the input.

$$\begin{aligned} \mathbb{P} &= \bigcup_{i\geq 0} DT(n^i) \\ \mathbb{NP} &= \bigcup_{i\geq 0} NT(n^i) \\ DEXT &= \bigcup_{i\geq 0} DT(2^{ni}) \\ NEXT &= \bigcup_{i\geq 0} NT(2^{ni}) \\ EXPTIME &= \bigcup_{i\geq 0} DT(2^{n^i}) \\ NEXPTIME &= \bigcup_{i\geq 0} NT(2^{n^i}) \end{aligned} \tag{2}$$

Theorem 3.5 (Complexity classes relations). *Complexity classes*

a) $NLOG \subseteq \mathbb{P}$

b) $PS = NPS$

c) $\mathbb{NP} \subseteq PS$

d) $PS = EXPTIME$

e) $NLOG \subsetneq PS \subsetneq EXPSPACE$

f) $\mathbb{P} \subsetneq DEXT \subsetneq EXPTIME$

Proof. a)

$$L \in NS(\log n) \overset{1.3}{\Rightarrow} \exists c_L : L \in DT(2^{c_L \log n}) = DT((2^{\log n})^{c_L}) = DT(n^{c_L}) \in \mathbb{P}$$

b) $PS \subseteq NPS$ trivial, as deterministic computation is a special case of non-deterministic.
$NPS \subseteq PS$

$$L \in NS \Rightarrow \exists i : L \in NS(n^i)$$

by Savic 1.11

$$L \in DS(n^{2i}) \Rightarrow L \in PS$$

c)

$$\forall L \in \mathbb{NP} \overset{1.1}{\Rightarrow} \exists i : L \in NT(n^i) \Rightarrow L \in DS(n^i) \Rightarrow \mathbb{NP} \subseteq PS$$

d)

$$L \in PS \Rightarrow \exists i : L \in DS(n^i) \subseteq NS(n^i) \overset{1.3}{\Rightarrow} \exists c_L : L \in DT(2^{c_L \log n}) \subseteq DT(2^{n^{i+1}}) \subseteq EXPTIME$$

e) by Savic 1.11

$$NS(\log n) \subseteq DS(\log^2 n)$$

by space hierarchy 1.7

$$\log^2 n \in o(n) \Rightarrow DS(\log^2 n) \subsetneq DS(n)$$

$$L \in PS \Rightarrow \exists i : L \in DS(n^i) \subseteq DS(2^n)$$

by space hierarchy 1.7

$$DS(2^n) \subsetneq DS(2^{2n})$$

f) by time hierarchy 1.10

$$\mathbb{P} \subseteq DT(2^n) \subsetneq DT(2^{2n}) \subseteq DEXT$$

as

$$2^n \log(2^n) = n2^n \in o((2^n)^2)$$

Then

$$DEXT \subseteq DT(2^{n^2}) \subsetneq DT(2^{n^3}) \subseteq EXPTIME$$

as

$$n^2(2^{n^2}) \in o(2^{n^3})$$

□

Note 3.6.

$$NLOG \subseteq \mathbb{P} \subseteq \mathbb{NP} \subseteq PS$$

Also

$$NLOG \subsetneq PS$$

Therefore one of the inclusions in first relation should be strict, or all of them. Still open question.
Moreover, if we prove strict equalities in leftmost and rightmost inclusions $\Rightarrow \mathbb{P} \neq \mathbb{NP}$.

3.3 Full Polynomial hierarchy

Definition 3.7 (Polynomial hierarchy). Consider a 3 sequences of classes

$$\Sigma_k, \Pi_k, \Delta_k$$

Where

1. $\Sigma_0 = \Pi_0 = \Delta_0 = \mathbb{P}$.
2. $\Sigma_{k+1} = \mathbb{NP}(\Sigma_k)$.
3. $\Pi_{k+1} = co - \mathbb{NP}(\Sigma_k)$.
4. $\Delta_{k+1} = \mathbb{P}(\Sigma_k)$.

And Polynomial hierarchy is:

$$PH = \bigcup_{i \geq 1} \Sigma_i^P (= \bigcup_{i \geq 1} \Pi_i^P = \bigcup_{i \geq 1} \Delta_i^P)$$

Definition 3.8 (Language complement). L is a language over alphabet τ

$$\overline{L} = \{x \in \tau^* |\ x \notin L\}$$

If $\mathcal{C}$ is a class of languages, then

$$L \in \mathcal{C} \iff \overline{L} \in co - \mathcal{C}$$

Theorem 3.9 (Polynomial Hierarchy). *Relations*

a) $\Sigma_1 = \mathbb{NP}$.

b) $\Pi_k = co - \Sigma_k \wedge \Sigma_k = co - \Pi_k$.

c) $\Sigma_{k+1} = \mathbb{NP}(\Pi_k)$.

d) $\Delta_{k+1} = \mathbb{P}(\Pi_k)$.

e) $\Pi_{k+1} = co - \mathbb{NP}(\Pi_k)$.

f) $\Sigma_{k+1} = \mathbb{NP}(\Delta_{k+1})$.

g) $\Pi_{k+1} = co - \mathbb{NP}(\Delta_{k+1})$.

Proof. a) $\mathbb{NP}(\mathbb{P}) = \mathbb{NP}$ also $\mathbb{P}(\mathbb{P}) = \mathbb{P}$.
Proof by embedding oracle computation via DTM simulation.
b) by definition, for $k \geq 1$

$$\Pi_k = co - \mathbb{NP}(\Sigma_{k-1}) = co - \Sigma_k$$

same for Σ_k by symmetry.
c) we can deduce $\overline{B}$ from questions to B by negating every request.
f) FIXME

$$\Sigma_{k+1} = \mathbb{NP}(\Sigma_k) \overset{\text{by i.h.}}{=} \overset{???}{\mathbb{NP}}(\Delta_{k-1}) = \mathbb{NP}(\mathbb{P}(\Sigma_k)) = \mathbb{NP}(\Delta_k)$$

Clearly

$$\mathbb{NP}(\Sigma_k) \subseteq \mathbb{NP}(\mathbb{P}(\Sigma_k))$$

as we made class of query languages larger.

$$L \in \mathbb{NP}(\mathbb{P}(\Sigma_k)) \Rightarrow \exists NTM\ M_n \exists E \in \mathbb{P}(\Sigma_k) : L = L(M_n, E)$$

Also

$$E \in \mathbb{P}(\Sigma_k) \iff \exists DTM\ M_d, \exists Q \in \Sigma_k : E = L(M_d, Q)$$

We want $L \in \mathbb{NP}(\Sigma_k)$. We proceed similarly as in $\mathbb{P}(\mathbb{P}) = \mathbb{P}$ by simulating M_n and every time it ask a query, use M_d to decide.
g)

□

Theorem 3.10 (Polynomial Hierarchy - 2). *Relations 2.*

a) $\Delta_k = co - \Delta_k$

b) $\mathbb{P}(\Delta_k) = \Delta_k$

c) $\Sigma_k \cup \Pi_k = \Delta_{k+1}$

d) $\Delta_k = \Sigma_k \cap \Pi_k$

e) if $\Sigma_k \subseteq \Pi_k \vee \Pi_k \subseteq \Sigma_k \Rightarrow \Pi_k = \Sigma_k$

Proof. a) for deterministic computation we can negate every answer. Specifically this rule means $\mathbb{P} = co-\mathbb{P}$
b) $\Delta_k \subseteq \mathbb{P}(\Delta_k)$ trivially. Since we can use same language as oracle with 1 query??
$\mathbb{P}(\Delta_k) \subseteq \Delta_k$. Proof by induction on k:

$$k = 0, \mathbb{P}(\mathbb{P}) = \mathbb{P}$$

$k \geq 1$ we have $\mathbb{P}(\Delta_{k-1}) \subseteq \Delta_{k-1}$ By definition $\mathbb{P}(\Delta_k) \subseteq \Delta_k$ is equivalent to

$$\mathbb{P}(\mathbb{P}(\Delta_{k-1})) \subseteq \mathbb{P}(\Delta_{k-1})$$

Both c) and d) use the fact, that deterministic computation is a special case of non-deterministic.
c) proof consists of 2 steps

1. $\Sigma_k \subseteq \Delta_{k+1} = \mathbb{P}(\Sigma_k)$
   Same argument as in first inclusion in b). Ask single query to the same language.
2. $\Pi_k \subseteq \Delta_{k+1} = \mathbb{P}(\Sigma_k)$
   Since we can negate queries
   $$\mathbb{P}(\Sigma_k) = \mathbb{P}(\Pi_k)$$
   Therefore using same argument as above with single query
   $$\Pi_k \subseteq \mathbb{P}(\Pi_k)$$

d) proof consists of 2 steps

1. $$\Delta_k \overset{def}{=} \mathbb{P}(\Sigma_{k-1}) \subseteq \mathbb{NP}(\Sigma_{k-1}) = \Sigma_k$$
2. $$\Delta_k \overset{\text{a)}}{=} co-\Delta_k = co-\mathbb{P}(\Sigma_{k-1}) = co-\mathbb{P}(\Pi_{k-1}) \subseteq co-\mathbb{NP}(\Pi_{k-1}) = \Pi_k$$

e) Assume $\Sigma_k \subseteq \Pi_k$ we need to prove reverse.

$$L \in \Pi_k \iff \overline{L} \in co-\Pi_k = \Sigma_k \xrightarrow{assumption} \overline{L} \in co-\Sigma_k = \Pi_k \iff L \in \Sigma_k$$

□

Theorem 3.11 (NP = co-NP). *if $A \in \mathbb{NP}$-complete $\wedge A \in co-\mathbb{NP}$-complete $\Rightarrow \mathbb{NP} = co-\mathbb{NP}$.*

Proof. Let $B \in \mathbb{NP}$ arbitrary. By the $\mathbb{NP}$-completeness of A $\exists DTM$ transducer M_t st

$$x \in B \iff M_t(x) \in A$$

$$A \in co-\mathbb{NP} \iff \overline{A} \in \mathbb{NP} \Rightarrow \exists NTM\ M_n : \overline{A} = L(M_n)$$

Also by the properties of polynomial reduction

$$x \in \overline{B} \iff M_t(x) \in \overline{A}$$

$\overline{A}$ can be recognized by M_n, therefore we can recognize $\overline{B}$ by NTM which is a concatenation of M_t and M_n. Therefore $\overline{B} \in \mathbb{NP} \iff B \in co-\mathbb{NP}$.
Same proof is valid for oracle TM. □

Theorem 3.12 (NP $\neq$ co-NP).

$$\mathbb{NP} \neq co-\mathbb{NP} \Rightarrow \mathbb{NP} \cup co-\mathbb{NP} \subsetneq \Delta_2$$

Proof. Let

$$L = \{(F, F') |\ F \in SAT \wedge F' \in \overline{SAT}\}$$

We can construct DTM with following algorithm for L

1. if($F \in SAT$)
2. if($\overline{F} \in SAT$)
3. reject
4. else
5. accept
6. else
7. reject

Therefore $L \in \mathbb{P}(SAT) \subseteq \Delta_2 = \mathbb{P}(\Sigma_1) = \mathbb{P}(\Pi_1)$.
Define

$$L_2 = \{(F, 0) |\ F \in SAT\} \subseteq L$$

where 0 is e.g. $(x \wedge \neg x)$.
Trivially $L_2 \simeq SAT$ therefore L contains $\mathbb{NP}$-complete language.
Now assume by contradiction $L \in co-\mathbb{NP}$. Using previous proof we can deduce $\mathbb{NP} = co-\mathbb{NP}$. □

3.4 Constrained quantifiers

Definition 3.13 (Constrained $\exists$).

$$\exists^{p(n)} x : R(x) := \exists x : |x| \leq p(n) \wedge R(x)$$

Definition 3.14 (Constrained $\forall$).

$$\forall^{p(n)} x : R(x) := \forall x : |x| \leq p(n) \wedge R(x)$$

Definition 3.15 ($\exists\mathcal{C}$). Let $\mathcal{C}$ be a class of languages, we define class of languages $\exists\mathcal{C}$ as following:

$$A \in (\exists\mathcal{C}) \overset{\text{def}}{\iff} \exists B \in \mathcal{C}, \exists p : x \in A \iff \exists^{p(|x|)} y : \langle x, y \rangle \in B$$

Note that if $\mathcal{C} = \mathbb{P} \Rightarrow \exists\mathbb{P} = \mathbb{NP}$. Since y is a branch of NTM (certificate)

Definition 3.16 ($\forall\mathcal{C}$). Let $\mathcal{C}$ be a class of languages, we define class of languages $\forall\mathcal{C}$ as following:

$$A \in (\forall\mathcal{C}) \overset{\text{def}}{\iff} \exists B \in \mathcal{C}, \forall p : x \in A \iff \forall^{p(|x|)} y : \langle x, y \rangle \in B$$

Note that if $\mathcal{C} = \mathbb{P} \Rightarrow \forall\mathbb{P} = co-\mathbb{NP}$.

Lemma 3.17 ($\exists\mathcal{C}, \forall\mathcal{C}, co-\mathcal{C}$). *Let $\mathcal{C}$ be an arbitrary class of languages. Then*

$$co-\exists\mathcal{C} = \forall(co-\mathcal{C}) \tag{3}$$

As

$$A \in (\exists\mathcal{C}) \iff \overline{A} \in (\forall(co-\mathcal{C}))$$

$$\mathcal{C} \subseteq \exists\mathcal{C} \wedge \mathcal{C} \subseteq \forall\mathcal{C} \tag{4}$$

Proof. Equation (3):
Negating the definition

$$A \in (\exists\mathcal{C}) \iff \exists B \in \mathcal{C}, \exists p : x \in A \iff \exists^{p(|x|)} y : \langle x, y\rangle \in B$$

we get

$$x \in \overline{A} \iff \forall^{p(|x|)} y : \langle x, y\rangle \in \overline{B}$$

Which is the same as

$$\overline{A} \in \forall(co-\mathcal{C}) \iff \exists\overline{B} \in co-\mathcal{C}, \exists p : \forall^{p(|x|)} y : \langle x, y\rangle \in \overline{B}$$

Equation (4):
For $\exists\mathcal{C}$ take $B = A \wedge y = \emptyset$. Clearly $\langle x, \emptyset\rangle \simeq x$.
For $\forall\mathcal{C}$ take polynomial with degree 0, only $y = \emptyset$ is accepted. □

Theorem 3.18 (Polynomial Hierarchy with quantifiers). *a)* $\exists\mathbb{P} = \mathbb{NP}$

b) $\forall\mathbb{P} = co-\mathbb{NP}$.

c) $\forall k > 0 : \exists\Sigma_k = \Sigma_k$.

d) $\forall k > 0 : \forall\Pi_k = \Pi_k$.

e) $\forall k \geq 0 : \exists\Pi_k = \Sigma_{k+1}$.

f) $\forall k \geq 0 : \forall\Sigma_k = \Pi_{k+1}$.

Proof. a) "$\exists\mathbb{P} \subseteq \mathbb{NP}$"

$$A \in (\exists\mathbb{P}) \iff \exists B \in \mathbb{P}, \exists p : x \in A \iff \exists^{p(|x|)} y : \langle x, y\rangle \in B$$

Construct an NTM that accepts A ($L(M) = A$):

1. guess y: $|y| \leq p(|x|)$
2. run deterministic verification $\langle x, y\rangle \in B$

B is in $\mathbb{P}$ therefore verification is deterministic. As $|y|$ is bounded by $p(|x|)$ total time complexity is polynomial.
"$\mathbb{NP} \subseteq \exists\mathbb{P}$"

$$A \in \mathbb{NP} \iff \exists NTM\ M : L(M) = A$$

Also

$$x \in A \iff \exists^{p(|x|)} y$$

where y encodes the accepting branch of M on input x.

// todo why??
Consequently any language B of pairs $\langle x, y\rangle$ is in $\mathbb{P}$. So, by definition

$$A \in \exists\mathbb{P}$$

b) using lemma lemma 3.17

$$\forall\mathbb{P} \xlongequal{\mathbb{P}=co-\mathbb{P}} \forall(co-\mathbb{P}) \xlongequal{lemma\ 3.17} co-\exists\mathbb{P} \xlongequal{a)} co-\mathbb{NP}$$

c) trivially $\Sigma_k \subseteq (\exists\Sigma_k)$. Now reversed.

$$A \in (\exists\Sigma_k) \iff \exists B \in \Sigma_k, \exists p : x \in A \iff \exists^{p(|x|)} y : \langle x, y\rangle \in B$$

We want

$$A \in \mathbb{NP}(\Sigma_{k-1}) = \Sigma_k$$

So, we construct an NTM M with oracle $C \in \Sigma_{k-1} : L(M, C) = A$, we also use

$$B \in \mathbb{NP}(\Sigma_{k-1}) \iff \exists M_B : B = L(M_B, C)$$

algorithm of M:

1. guess y: $|y| \leq p(|x|)$
2. run M_B on $\langle x, y\rangle$.

Note that

$$\exists y \exists z \iff \exists\langle y, z\rangle$$

d)

$$\forall\Pi_k = \forall(co-\Sigma_k) \xlongequal{lemma\ 3.17} co-\exists\Sigma_k \xlongequal{c)} co-\Sigma_k = \Pi_k$$

Note that now we cannot prove e)
f) follows from e)

$$\forall\Sigma_k = \forall(co-\Pi_k) \xlongequal{lemma\ 3.17} co-\exists\Pi_k \xlongequal{e)} co-\Sigma_{k+1} = \Pi_{k+1}$$

□

Lemma 3.19 (A^*). *Let $\mathcal{C}$ be an arbitrary class from PH. Then*

$$\forall A \in \mathcal{C} \iff A^* \in \mathcal{C}$$

Where

$$A^* = \{x|\ \exists n \exists y_1 \in A \ldots \exists y_2 \in A : x = (y_1, \ldots, y_n)\}$$

And $\forall$ is a normal quantifier.
In other words, concatenation of the words from A are also in A.
Note that comma separators are important to stay in the same complexity class. We cannot guess the split by brute force.

Proof. Separately for every class in PH.
1) $C = \Sigma_0 = \Pi_0 = \Delta_0 = \mathbb{P}$.
$\Leftarrow$ trivially $A \subseteq A^*$ for $n = 1$. $x = (y)$.
"$\Rightarrow$". $\exists DTM\ M : A = L(M)$
We run M on all $y_1, y_2, \ldots, y_n$. We accept $\iff$ all computations on y_i accepts.
2) $C = \Delta_k$. Same as in 1). We have DTM with an oracle TM.
3) $C = \Sigma_k, k > 0$. We have $\exists NTM\ M$ with oracle $D \in \Sigma_{k-1} : A = L(M, D)$. We simulate computation of M on y_1. In every accepting path, we run M on y_2. And so on.
4) $C = \Pi_k$ If there is at least one $y_i \in co - A$, whole string $(y_1, y_2, \ldots, y_n) \in (co - A)^*$. Therefore we proceed roughly the same as in 3), but we proceed for rejecting leaves. □

Consequence 3.20. *If we have $A, B, T \in \mathcal{C} \Rightarrow D \in \mathcal{C}$ where*

$$D = \{x |\ \exists a \in A, \exists b \in B, \exists t \in T : x = (a, b, t)\}$$

Proof is the same, as of the lemma, but we have multiple TM.

Theorem 3.21 (Polynomial hierarchy consequences).

$$\exists \Pi_k = \Sigma_{k+1}$$

e) from Polynomial Hierarchy with quantifiers 3.18

Proof. "$\exists \Pi_k \subseteq \Sigma_{k+1}$"

$$A \in \exists \Pi_k \iff \exists B \in \Pi_k \exists p : x \in A \iff \exists^{p(|x|)} y : (x, y) \in B$$

Construct an $NTM\ M$ which works as following:

1. read x
2. guess $y : |y| \leq p(|x|)$.
3. ask oracle if $(x, y) \in B$

We get

$$A = L(M, B) \Rightarrow A \in \mathbb{NP}(\Pi_k) = \mathbb{NP}(\Sigma_k) \stackrel{def}{=} \Sigma_{k+1}$$

"$\exists \Pi_k \supseteq \Sigma_{k+1}$"
Proof by induction on k.
$k = 0$.

$$\Sigma_1 \subseteq \exists \Pi_0 = \exists \mathbb{P} = \mathbb{NP}$$

induction hypothesis: $\Sigma_k \subseteq \exists \Pi_{k-1}$. Let $A \in \Sigma_{k+1}$ be arbitrary. Then

$$\exists NTM\ M \exists B \in \Sigma_k : A = L(M, B)$$

$x \in A \iff \exists$ accepting computation (poly long) of M on input x which asks if $z_1, z_2, \ldots, z_n \in B$ and if $w_1, \ldots, w_n \in co - B$. Alternatively

$$x \in A \iff \exists^{p(|x|)} y, \exists^{p(|x|)} z = (z_1, z_2, \ldots, z_n), \exists^{p(|x|)} w = (w_1, \ldots, w_n)$$

s.t. y encodes an accepting computation of M on X with positive queries $z_1, \ldots, z_n$ and negative queries $w_1, \ldots, w_n$.
We claim that the language L of pairs $(x, y) \in \mathbb{P} \subseteq \Pi_k$.

We know

$$z_i \in B^* \Rightarrow z \in B^* \in \Sigma_k \overset{i.h.}{\subseteq} \exists\Pi_{k-1}$$

$$w_i \in (co-B)^* \Rightarrow w \in B^* \in \Sigma_k \overset{i.h.}{\subseteq} \exists\Pi_{k-1}$$

So

$$x \in A \iff \exists^{p(|x|)} y \exists^{p(|x|)} z \exists^{p_1(|z|)} y \exists^{p(|x|)} w$$

□

Definition 3.22 (PH by alternating quantifiers).

$$A \in \Sigma_k \iff \exists B \in \mathbb{P} \exists p : x \in A \iff \exists^{p(|x|)} y_1, \forall^{p(|x|)} y_2 \ldots : (x, y_1, y_2, \ldots, y_k) \in B$$

Also

$$A \in \Pi_k \iff \exists B \in \mathbb{P} \exists p : x \in A \iff \forall^{p(|x|)} y_1, \exists^{p(|x|)} y_2 \ldots : (x, y_1, y_2, \ldots, y_k) \in B$$

Proof. Proof by induction on k.
For $k = 0$ we have no quantifiers.

$$A \in \mathbb{P} \exists B \in \mathbb{P}$$

we can take $A = B$.
$k \to k+1$. Let $A \in \Sigma_{k+1} = \exists\Pi_k$. Then

$$\exists\Pi_k \exists p : x \in A \iff \exists^{p(|x|)} y : (x, y) \in B$$

By the i.p.

$$\Rightarrow (x, y) \in B \iff \forall^{p(|x|)} y_1, \exists^{p(|x|)} y_2 \ldots : ((x, y), y_1, y_2, \ldots, y_k)$$

□

Example 3.23. Language from Σ_2. Optimization version (Boolean minimization).
Input: CNF F
Output: CNF $H : H \equiv F \wedge |H|$ is minimal (we can define minimal in many ways, e.g. minimal in bits to represent).
Now we convert into decision problem.
Input: CNF $F, k \in \mathbb{N}$
Question: $\exists CNF\ H : F \equiv H \wedge |H| \leq k$.
Problem is in Σ_k since

$$F \in BM \iff \exists^{|H| \leq |F|} H, \forall (x_1, \ldots, x_n) : H \leq k \wedge F(x_1, \ldots, x_n) = H(x_1, \ldots, x_n)$$

We also know $BM \in \Sigma_2 - complete$.

Note 3.24. If we have a hard problem, it can be encoded in CNF and solved by CNF machinery. CNF solvers are highly optimized over more than 20 years.

3.5 PH collapse

Theorem 3.25 (Polynomial hierarchy collapse at level k). *if* $\Sigma_k = \Pi_k$ *for some* $k > 0$ *then*

$$\forall j \geq 0 : \Sigma_{k+j} = \Pi_{k+j} = \Sigma_k$$

Proof. By induction on j. For 0 is true by assumption.
Induction step:

$$\Sigma_{k+j+1} \overset{3.18\ c)}{=\!=} \exists \Pi_{k+j} \overset{I.H.}{=\!=} \exists \Sigma_{k+j}$$

By 3.18 c)

$$\exists \Sigma_{k+j} = \Sigma_{k+j}$$

By I.H. again

$$\Sigma_{k+j} = \Sigma_k$$

Similarly for Π_{k+j+1}.

$$\Pi_{k+j+1} \overset{3.18\ f)}{=\!=} \forall \Sigma_{k+j} \overset{I.H.}{=\!=} \forall \Pi_{k+j} \overset{I.H.}{=\!=} \Sigma_k$$

□

Theorem 3.26 (PH not collapse). *Either* $\forall k : \Sigma_k \subset \Sigma_{k+1}$ *or PH collapses.*

Proof. Assume

$$\Sigma_k = \Sigma_{k+1}$$

We know

$$\Sigma_{k+1} = \mathbb{NP}(\Sigma_k) = \mathbb{NP}(\Pi_k) \supset \Pi_k$$

Implies by assumption

$$\Pi_k \subseteq \Sigma_k \Rightarrow \Sigma_k = \Pi_k$$

Then by 3.25 PH collapses after k.
In particular for $k = 0$ we get

$$\mathbb{P} = \mathbb{NP} \Rightarrow PH = \mathbb{P}$$

□

Consequence 3.27.

$$\textit{If } \exists k \in \mathbb{N} : \mathbb{P} = \Sigma_0 \subset \Sigma_k \Rightarrow \mathbb{P} \subset \mathbb{NP}$$

Proof. By reversing previous condition. □

Definition 3.28 (PSPACE-complete). L is PSPACE-complete if:
$L \in PSPACE$ and
$\forall L_a \in PSPACE : L_a$ is poly time reducible to L.
Note that we use Time reducibility for Space class.

Lemma 3.29 ($PH = \Sigma_k$). *Let* L *be PS-complete and* $L \in \Sigma_k$ *then* $PH = \Sigma_k$.

Proof. Take $L_2 \in PS$ arbitrary, then by the polynomial reduction

$$\exists DTM\ M_d : x \in L_2 \iff M_d(x) \in L$$

M_d is a transducer.
Also there is acceptor

$$\exists NTM\ M_n, \exists D \in \Sigma_{k-1} : L = L(M_n, D)$$

Then we construct new NTM by concatenation of M_d and M_n.

$$L_2 \in PS$$

Therefore

$$PS \subseteq \Sigma_k$$

We already know that

$$PH \subseteq PS$$

Therefore

$$PH = \Sigma_k$$

□

Consequence 3.30. *if $PH = PS$ then*

$$\exists k \in \mathbb{N} : PH = \Sigma_k$$

Assuming that $\exists L \in PS$-complete.
Which implies, that if PH grows infinitely and no PS-complete is in PH. Then $PS \setminus PH$ contains all PS-complete languages.

Proof. We take L, by $PH = PS$

$$\exists k : L \in \Sigma_k$$

then by lemma 3.29

$$PH = PS$$

□

4 PS-complete lang

Definition 4.1 (QBF - quantifiable boolean formula).

1. if x is a variable then x is a QBF and x is a *free* variable
2. if E_1, E_2 are QBF then

   $$\neg E_1, (E_1) \wedge (E_2), (E_1) \vee (E_2)$$

   are also QBFs.

   And the status of variables (free/bounded) does not change.
3. if E is a QBF then

   $$\exists x(E), \forall x(E)$$

   are also QBFs. And all ocurrences of x become bounded. Status of other variables does not change.

Definition 4.2 (QBF problem). QBF problem (language).
Input: QBF F with no free variables.
Question: $F = 1$??

How do we evaluate QBF with no free variables?

- $\exists x(E) \iff E_0 \vee E_1$

- $\forall x(E) \iff E_0 \wedge E_1$

Where E_0 is formula where every occurrence of x is replaced by 0. Similarly E_1.

Example 4.3.

$$\forall x(\forall x(\exists y(x \vee y)) \wedge \neg x)$$

by rules above

$$(\forall x(\exists y(x \vee y)) \wedge \neg 0) \wedge (\forall x(\exists y(x \vee y)) \wedge \neg 1)$$

Note 4.4. SAT - language of satisfiable CNFs. We can think of it as

$$\exists x_1 \exists x_2 \ldots \exists x_n(F(x_1, x_2, \ldots, x_n))$$

Therefore SAT is a special case of QBF.

Theorem 4.5 (QBF $\in$ PS). *QBF $\in$ PS.*

Proof. We construct DTM to evaluate QBF without free variables as following

- $\neg(E) \rightarrow$ evaluate E and negate all results
- $(E_0) \vee (E_1) \rightarrow$ evaluate E_0, E_1 then by disjunction
- $(E_0) \wedge (E_1) \rightarrow$ evaluate E_0, E_1 then by conjunction
- $\exists(E) \rightarrow$ compute E_0, E_1, then compute $E_0 \vee E_1$
- $\forall(E) \rightarrow$ compute E_0, E_1, then compute $E_0 \wedge E_1$

We have at most n operators. We get binary tree that evaluates the formula. Where every branch is bounded by total length of initial formula.
$\mathcal{O}(n^2)$ is enough space for evaluation. □

Example 4.6.

$$F = \vee_{1 \leq j \leq n}(x_i \rightarrow y_i) = \vee_{1 \leq j \leq n}(\neg x_i \vee y_j)$$

Can be viewed as bipartite graph.
Claim: The resulting formula is shortest CNF representing F. By the completness of resolution. Length changed to $\Theta(n^2)$.
Now 2nd formula

$$H = (\exists z)[(\wedge_{1 \leq i \leq n}(x_i \rightarrow z)) \wedge (\wedge_{1 \leq j \leq n}(z \rightarrow y_j))]$$

H is an encoding of F with auxiliary variables. Can be viewed as bipartite graph but with single node in between parts.
However, H is shorter since it is $\Theta(n)$.

Proof. As every x implies every y models of F are:

$$(0, 0, 0, \ldots, *, * \ldots, *) \cup (*, * \ldots, *, 1, 1, \ldots, 1)$$

Where $*$ represents arbitrary value.
If we rewrite H and substitute $0 \vee 1$ for z we get:

$$[(\wedge_{1 \leq i \leq n}(x_i \rightarrow 0)) \wedge (\wedge_{1 \leq j \leq n}(0 \rightarrow y_j))] \vee [(\wedge_{1 \leq i \leq n}(x_i \rightarrow 1)) \wedge (\wedge_{1 \leq j \leq n}(1 \rightarrow y_j))]$$

Therefore models are the same. □

Note 4.7. Trick with auxiliary variables is used, if we have a requirement of only one x_i to be 1. Which can be represented by formula:

$$\wedge_{1\leq i,j\leq n}(x_i \vee \neg x_j)$$

Which is $\Theta(n^2)$ and auxiliary variable makes it linear.

Theorem 4.8 (QBF is PS-hard). *QBF is PS-hard (sketch).*

Proof. Every $L \in PS$ arbitrary can be reduces to QBF in poly time.
By the definition of PS

$$\exists DTM\ M : L = L(M), \exists p(n) \text{ M accepts L in space } p(n)$$

We have $2^{c_m p(n)}$ configurations of M and every configuration can be encoded by string of length

$$c_m p(n) = m(n) := m$$

We assume, that there is only 1 accepting configuration.
$x \in L \iff \exists$ path of length m in *configuration graph* from $C_0 \to Cacc$. Use similar algorithm as in Savic theorem 1.11, but encode computation in QBF.
Notation, where φ is an encoding of allowed transition in Cook-Levin theorem. We construct QBF ψ.

- $\psi_0(C, C') = 1 \iff \varphi_m(C, C')$ is satisfiable
- $\psi_i(C, C') = 1 \iff$ there exists path $C \to C'$ of length 2^i
- $\psi_m(C_0, C_{acc}) = 1 \iff x \in L$

Obvious idea that would not work

$$\psi_i(C, C') = \exists C_{int}(\psi_i(C, C_{int}) \wedge \psi_i(C_{int}, C'))$$

Since every such change doubles size of the formula, we end up with

$$|\psi_m| \in \Omega(2^m p(n))$$

Main idea

$$\psi_i(C, C') = \exists C_{int} \forall D_1, D_2[(D_1 = C \wedge D_2 = C') \vee (D_1 = C_{int} \wedge D_2 = C')] \Rightarrow \psi_{i-1}(D_1, D_2)$$

Formally, implication could be replaced by $\neg x \vee y$. Now

$$|\psi_i| = |\psi_{i-1}| + \mathcal{O}(m)$$

Therefore

$$|\psi_m| = \mathcal{O}(m^2)$$

Also, going from $\psi_{i-1} \to \psi_i$ we need 1 existential, 1 universal quantifier. In the end, ψ_m has m pairs of alternating existential and universal quantifiers.
Therefore $QBF \in \Sigma_m$. □

4.1 P-completeness

Note 4.9. If we use polynomial time reducibility, almost all languages(except trivial: empty and all words) are $\mathbb{P}$-complete.
Therefore we use a different reducibility.

Definition 4.10 (log-space reducibility). A is *log-space* reducible to B if $\exists$ DTM transducer M that works in log space (excluding input and output tape). Such that $x \in A \iff M(x) \in B$.

Definition 4.11 ($\mathbb{P}$-complete). L is $\mathbb{P}$-complete $\iff L \in \mathbb{P} \wedge \forall A \in \mathbb{P}\, A$ is log-space reducible to L.

Theorem 4.12 (P-complete vs LOG). *Let L be $\mathbb{P}$-complete and $L \in LOG = DS(\log(n)) \Rightarrow \mathbb{P} = LOG$.*

Proof. Since $c_n^{\log n} = (2^{\log n})^{\log c_n} = n^{\log c_n}$.

$$LOG \subseteq \mathbb{P}$$

We want

$$\mathbb{P} \subseteq LOG$$

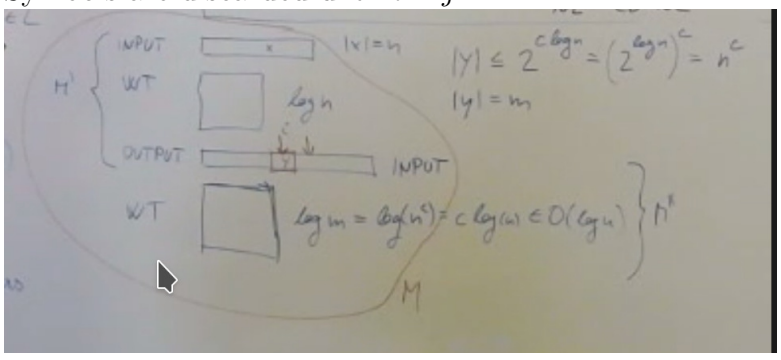
Let $B \in \mathbb{P}$ arbitrary, we need log-space acceptor for B $\Rightarrow B \in LOG$. From L is $\mathbb{P}$-complete $\Rightarrow \exists$ log-space DTM $M_L : x \in B \iff M_L(x) \in L$. From $L \in LOG \Rightarrow \exists$ log-space DTM acceptor $M_{log} : L = L(M_{log})$.
We cannot simply concatenate 2 machines, as output tape of the first machine M_L becomes work tape of the 2nd. Output tape is not guaranteed to be log-space. Let Y be the output of M_L

$$|Y| \leq 2^{c_M \log n} = n^{c_M}$$

Idea: keep just current symbol on output of M_L and the position. Then start the next step of M_{log}. Then restart M_L and discard output with position $< i$. Repeat.
Works, because we do not worry about Time, but space.
We need 2 counters $i, j \in \{1, \ldots, |Y|\}$. Which require

$$\log(n^c) = c \log n$$

Where i keeps the position, updates with every move of the head. j is reset to 0 and is incremented with every symbol M_{log} outputs.
Symbols are discarded until $i = j$.



□

Consequence 4.13. *Let L be $\mathbb{P}$-complete and $L \in NLOG = NS(\log(n)) \Rightarrow \mathbb{P} = NL$.*
Proof is same, but acceptor is non deterministic.

Q: what if we use log-space reducibility in $\mathbb{NP}$-complete definition? This is stricter, since if we can reduce in log-space, we can also reduce in polynomial time (by time and space comparison with 2^n).

4.2 $NL = co-NL$ by Szelepcsenyi-Immerman

Theorem 4.14 (Szelepcsenyi-Immerman(1988)).

$$NL = co-NL$$

Proof. Similar proof as in case $L \in \mathbb{NP}$-complete $\wedge L \in co-\mathbb{NP}$. But the reducibility is again log space definition 4.10.
Assume, $L \in NL$-complete. Let $L_a \in NL$ be arbitrary.
Let M_{log} be a log space transducer, M be acceptor for $co-L$.
Then $M_{log} + M$ is an acceptor for $co-L_a$.
Output of the transducer can be quite large. Which then becomes working tape for acceptor. Similarly, as in previous theorem 4.12 output of the transducer is added character by character.
Let the desirable language be

$$L = PATH = \{(G,s,t)|\text{ G is an undirected graph in which } \exists Path(s,t)\}$$

Graph can be encoded by the following:

- for n vertices $\log n$ counter is enough.
- for each vertex we store list of neighbors.

Easy to see, that PATH $\in \mathbb{P}$, since BFS, DFS can solve problem in poly time.
PATH $\in LOG$ is an open question. However, with NTS log space is enough since we can try all paths with size n by picking the next neighbor randomly. In each step it is sufficient to remember code of the current vertex and path size counter on separate tapes. Both of them are of size $\mathcal{O}(\log n)$.
The algorithm:

1. vertex curr $= s; counter = 0;$
2. while$(counter < n)\{$
3. pick neighbor U at random; if$(U = t)$ accept;
4. $curr = U;$
5. $\}$
6. reject.

To show that PATH is NL-hard, we will treat the graph as the configuration graph. And the $Path(s,t)$ will be a path from C_0 to C_{accept}. WLOG there is only one C_{accept} configuration.
Let $L_a \in NL$ be arbitrary, therefore $\exists$ log space acceptor $M : L_a = L(M)$.
Transducer M_{log} encodes the configuration graph of L_a.
Code of the acceptor is not a problem, as size of M does not depend on the input. Transition table if M is embedded in the control unit.
Space needed for 1 configuration of M is $\mathcal{O}(\log n)$. As the work tape is bounded by the assumption, and for the position of the head $\mathcal{O}(\log n)$ is also enough.
The last step is to prove co-PATH $\in NL$. Algorithm is the following:

- count vertices reachable from s in G.

- count vertices reachable from s in $G \setminus \{t\}$.
- if equal - accept.

There is no need to generate encoding of graph $G \setminus \{t\}$, since t is part of input and we can ignore it by single "if".
How to count reachable vertices? The number is bounded by n therefore, $\mathcal{O}(\log n)$ space is enough.

Definition 4.15 (R_i).

$$R_i = \{u|\ u \text{ is reachable from } s \text{ by path length } \leq i\}$$

Also

$$R_i = R_{i-1} \cup \{a|\ \exists b \in R_{i-1} : (a,b) \in E(G)\}$$

We want to compute $|R_n|$. To do so, we compute non-deterministically compute $|R_0|, |R_1|, \ldots, |R_n|$. At each step, we remember only the last number. Algorithm to compute R_i:

1. $|R_0| = 1$;
2. guess $0 \leq g \leq n$ and verify:
   (a) $g \geq |R_i|$
   Generate non-deterministically all subsets $V \subseteq V(G)$ of vertices of size g. Vertices are ordered by increasing number of their codes. Check whether $\forall v \in V : \exists Path(s,v) : |Path(s,v)| = i$.
   (b) $g \leq |R_i|$.
   Is equivalent to check
   $$|V \setminus R_i| \geq n - g$$
   Algorithm:
   i. non deterministically generate $V_t \subseteq V(G) : |V_t| = (n-g)$ vertices.
   ii. $\forall a \in V_t$ check $a \in V(G) \setminus R_i$.
   iii. generate $V_r \subseteq V(G) : |V_r| = |R_{i-1}|$ non deterministically: select $|R_{i-1}|$ vertices and check $Path(s,a) : |Path(s,a)| = i-1$. As R_{i-1} is unique we get same set every time.
   iv. $\forall b \in V_r$ check $a \neq b \wedge (b,a) \notin E(G)$.

Required space is again $\mathcal{O}(\log n)$ as in each step algorithm stores constant number of vertices. □

5 Non-uniform computation

Definition 5.1 (Uniform computation). Uniform models of computation - single algorithm for all inputs. (DTM, NTM, RAM ...)

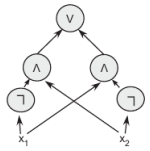
Definition 5.2 (Non-uniform computation). Algorithm may vary for different input lengths. However, inputs of the same lengths are handled by the same algorithm.
For example Boolean circuits.

Definition 5.3 (Boolean circuit). Boolean circuit with n inputs (and single output) is an acyclic directed graph with n source virtices. Source vertex has $deg_{in}(s) = 0$. And single sink vertex $deg_{out}(t) = 0$.
Source vertices represents the inputs, sink - output. All other vertices are gates (NOT, AND, OR).
AND, OR gates have 2 inputs (in degree). NOT has only 1 input.
Size of the circuit is # of vertices.

Note 5.4. Because of the selected elementary gates, number of edges is bounded by $2n$.

Example 5.5 (XOR gate). $x_1 XOR x_2$ as a DNF formula:

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$



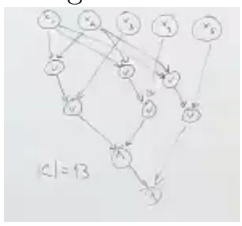
Source [1].
Or CNF

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$$

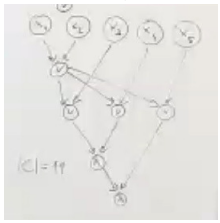
Example 5.6.

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5)$$

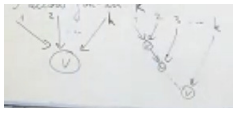
The gate will be the following:



Size of the gate is $|C| = 13$.
However, if we change the definition and allow multiple outputs, size is $|C| = 11$.



Observation 5.7. If we allow gates with k inputs or outputs, the equivalent binary circuit will have $(k-1)$ gates. Depth of the tree can be optimized by balancing.



Note 5.8. Having gates with unlimited inputs can result in an exponential blowup.

Q: do we restrict the gates to binary? A: no, but we allow only finite?

Note 5.9. Circuit with input of size n can be viewed as an device that recognizes Language of binary words of length n.
A sequence $\{C_n\}$ of circuits of various length is a device that recognizes some Language. Enormous switch by n.

Notation 5.10. For circuit C and input vector $x : C(x)$ is a value of the output gate.

Definition 5.11 (Family of circuits). Let T be a function. A family of circuits of size $T(n)$ is a sequence of circuits $\{C_n\}_{n\in\mathbb{N}}$. Where

$$\forall n \in \mathbb{N} : C_n \text{ has n inputs } \&|C_n| \leq T(n)$$

Language L is in class SIZE($T(n)$) is there exists a family of circuits $\{C_n\}$ of size $T(n)$ such that

$$\forall x \in L \iff C_n(x) = 1$$

Q: T is a boolean function?

Example 5.12.

$$L = \{1^n |\ n \in \mathbb{N}\}$$

Then $L \in$ SIZE($\mathcal{O}(n)$).
The circuit is a conjunction of all inputs.

Example 5.13.

$$L = \{(i,j,i+j) |\ i,j \in \mathbb{N}\}$$

Then $L \in$ SIZE($\mathcal{O}(n)$).
The circuit performs binary addition of i and j, then compare with $i+j$. Binary addition can be done by circuit of linear size.

Definition 5.14 ($\mathbb{P}$/poly class).

$$\mathbb{P}/\text{poly} = \bigcup^{\infty} SIZE(n^i)$$

Class of languages recognizable by families of circuits of polynomial size.

Definition 5.15 (Oblivious DTM). IF L is recognizable by DTM M in time $p(n)$ then it is also recognizable by DTM M_1 with single work tape with time complexity $p^2(n)$. Moreover, the head movement of M_1 are independent of the contents of its tapes but only on the input length (M_1 always performs a sequence of left to right and back sweeps of the same form regardless of what is the input).
Source [1, p. 37].

Theorem 5.16 ($\mathbb{P} \subseteq \mathbb{P}$/poly). $\mathbb{P} \subseteq \mathbb{P}$/poly.

Proof. Idea: for fixed input length n TM can be simulated by polynomial size circuit. The only problem is to get the symbol from work/input tape without storing the whole tape. Which is solved by the previous configuration with current symbol.
Let $L \in \mathbb{P}$ be arbitrary. WLOG there is an oblivious DTM M definition 5.15 such that $L = L(M)$. M works in time $T(n)$, where T is a polynomial. WLOG $T(n)$ is the exact time M needs for computation. We can choose $T(n)$ by trying different polynomials in parallel with M using the fact, that polynomials are *time constructible*.

Knowing that TM did exactly $T(n)$ steps tells us that there were $T(n)$ displays (state, input symbol, work tape symbol).
Let $|x| = n$ be input, let

$$d_1, d_2, \ldots, d_{T(n)}$$

be binary string encoding displays during the computation of M on x.
Since M is *oblivious*, the position of the head i uniquely defines the position i_v, i_p. Because the position of the head is uniquely determined by the # of steps.
i_v, i_p are the display indexes the last time the input head was in the same position and the work head was in the same position as in d_i. The gates representing the displays d_{i_v} and d_{i_p} are the inputs to the gate C_i. It can also happen, that next symbol under the head has not been seen during the computation yet. For such case, we add one more input x_j - input bit.
Similarly, i_p should not be defined. However, we assume that work tape is blank before the computation and symbol is also blank.
We should add one more gate that check whether TM finished in accepting state. Outputs 1 if was in accepting state and 0 o/w.
Complexity: $|C_i| = \mathcal{O}(1) \Rightarrow |C| = \mathcal{O}(T(n))$. □

Corollary 5.17. *The family of circuits $\{C_n\}$ from previous theorem not only exists but also can be efficiently constructed. Meaning there exists DTM M which on the input 1^n (only marks the length) outputs a ... of C_n. Moreover M works in polynomial time and Log space.*
The only non-trivial thing in circuit construction is to compute indexes i_v, i_p from i and n. M outputs the circuit C_i in $\mathcal{O}(\log n)$ time and space.
Space is reused therefore total space complexity is also $\mathcal{O}(\log n)$. However, having $T(n)$ gates time complexity is $\mathcal{O}(T(n) \cdot \log n)$ which is polynomial.

Note 5.18 ($\mathbb{P} \not\supseteq \mathbb{P}/\text{poly}$). $\mathbb{P} \supseteq \mathbb{P}/\text{poly}$ is not true since every unary language

$$L \subseteq \{1^n |\ n \in \mathbb{N}\}$$

is in $\mathbb{P}/\text{poly}$.
If $1^n \in L \Rightarrow C_n$ is a tree of $\wedge$. Otherwise, C_n outputs 0.
On the other side

$$UHALT = \{1^n |\ n \text{ is a Godel number of } \langle M, x\rangle : M(x)\downarrow\}$$

is not in $\mathbb{P}$ as it is a variant of Halting problem which is undecidable.
Q: how can boolean circuit recognize UHALT? How to detect that that TM halts in this case?

Definition 5.19 ($\mathbb{P}$-uniform family circuits). A family of circuits $\{C_n\}$ is $\mathbb{P}$-uniform if there exists DTM M which on input 1^n outputs the description of $\{C_n\}$ and works in polynomial time.

Theorem 5.20 ($\mathbb{P}$-uniform family circuits). *L is accepted by $\mathbb{P}$-uniform family of circuits $\iff$ $L \in \mathbb{P}$.*

Proof. "$\Rightarrow$". For an input x DTM M will simulate DTM M_g which outputs C_n on 1^n, guaranteed by uniformity. Then M simulates C_n on x.

$$x \in L(M) \iff C_n(x) = 1$$

"$\Leftarrow$". Follows from the corollary corollary 5.17. □

Definition 5.21 (Log Space uniform family circuits). A family of circuits $\{C_n\}$ is $\mathbb{P}$-uniform if there exists DTM M which on input 1^n outputs the description of $\{C_n\}$ and works in Log space.

Theorem 5.22 (Log Space uniform family circuits). *L is accepted by Log Space uniform family of circuits $\iff L \in \mathbb{P}$.*

Proof. "⇒". For an input x DTM M will simulate DTM M_g which outputs C_n on 1^n, guaranteed by uniformity. Then M simulates C_n on x.

$$x \in L(M) \iff C_n(x) = 1$$

The only difference between current theorem and previous is an assumption on M_g. However, TM that works in Log space also works in polynomial time.
"⇐". Follows from the corollary corollary 5.17. □

5.1 Cook-Levin alternative proof

Definition 5.23 (CRT-SAT). CRT-SAT is a language of binary strings that encode boolean circuits which for *some* input output 1.

$$CRT - SAT = \{C |\ \exists n : C(n) = 1\}$$

Lemma 5.24 (CRT-SAT $\in \mathbb{NP}$). *CRT-SAT $\in \mathbb{NP}$.*

Proof. The certificate is the input for which $C(n) = 1$. Check by simulating circuit using DTM. □

Lemma 5.25 (CRT-SAT $\in \mathbb{NP}$-hard). *CRT-SAT $\in \mathbb{NP}$-hard.*

Proof. Let $L \in \mathbb{NP}$ arbitrary.

$$L \in \mathbb{NP} \iff \exists DTMM : x \in L \iff \exists b \in \{0,1\}^{p(n)} : M(x,b) = 1$$

M works in polynomial time, b encodes the accepting branch of the NTM computation. From $\mathbb{P} \subseteq \mathbb{P}/\text{poly}$ 5.16 exists $\{C_n\}$ which recognizes the same language as M. For the pair (x,b) of size $n + T(n)$ we have circuit C_n:

$$C_n(x,b) = 1 \iff M(x,b) = 1$$

For fixed x we construct a circuit $C_n^x \in C_n$ by fixing the inputs of C_n to x. Therefore C_n^x has single input b.

$$x \in L \iff \exists b : M(x,b) = 1 \iff \exists b : C_n(x,b) = 1 \iff \exists b : C_n^x(b) = 1 \iff C_n^x \in CRT - SAT$$

□

Consequence 5.26 (CRT-SAT $\in \mathbb{NP}$-complete). *CRT-SAT $\in \mathbb{NP}$-complete, follows from 2 previous lemma.*

Theorem 5.27 (Cook-Levin alternative proof).

Proof. Idea: $L \in \mathbb{NP} \to CRT - SAT \to 3 - SAT$.
Let C be a circuit with input vertices $x_1, \ldots, x_n$ and gates $g_1, \ldots, g_n$. Where g_n is an output gate. The next step is to identify vertices and gates with binary variables.
Known as Tseitin encoding.

- if g_i is NOT with single input g_j then formula is XOR

$$(g_i \vee g_j) \wedge (\neg g_i \vee \neg g_j)$$

- if g_i is AND with inputs g_j, g_k then formula consists of 2 implications

$$(g_i \Rightarrow (g_j \wedge g_k)) \wedge ((g_j \wedge g_k) \Rightarrow g_i)$$

Then convert to CNF

$$(g_i \Rightarrow (g_j \wedge g_k)) = g_i \Rightarrow g_j \wedge g_i \Rightarrow g_k = (\neg g_i \vee g_k) \wedge (\neg g_i \vee g_k)$$

The second part

$$((g_j \wedge g_k) \Rightarrow g_i) = (\neg g_j \vee \neg g_k \vee g_i)$$

Altogether

$$(\neg g_i \vee g_k) \wedge (\neg g_i \vee g_k) \wedge (\neg g_j \vee \neg g_k \vee g_i)$$

- if g_i is OR with inputs g_j, g_k then formula consists of 2 implications:

$$(g_i \Rightarrow (g_j \vee g_k)) \wedge ((g_j \vee g_k) \Rightarrow g_i)$$

Then convert to CNF

$$(g_i \Rightarrow (g_j \vee g_k)) = (\neg g_i \vee g_j \vee g_k)$$

The second part

$$((g_j \vee g_k) \Rightarrow g_i) = g_j \Rightarrow g_i \wedge g_k \Rightarrow g_i = (\neg g_j \vee g_i) \wedge (\neg g_k \vee g_i)$$

Altogether

$$(\neg g_i \vee g_j \vee g_k) \wedge (\neg g_j \vee g_i) \wedge (\neg g_k \vee g_i)$$

- unit clause (g_n).

□

6 TM with advice

Definition 6.1 ($DT(T(n))|_{a(n)}$). Let $T, s : \mathbb{N} \to \mathbb{N}$. Then

$$DT(T(n))|_{a(n)}$$

is a class of languages recognizable by DTM in time $T(n)$ with $s(n)$ bit advice α_n.

$$L \in DT(T(n))|_{s(n)} \iff \exists DTMM \& \exists \{\alpha_n\}_{n\in\mathbb{N}}, \alpha_n \in \{0,1\}^{s(n)} :$$

$$\forall x \in \{0,1\}^n : x \in L \iff M(x, \alpha_n) = 1$$

And M makes $\mathcal{O}(T(n))$ steps on the input (x, α_n).

Note 6.2. TM with advice is an non-uniform computation as advice size depends on input size.

Theorem 6.3 ($\mathbb{P}$/poly). $\mathbb{P}/\text{poly} = \bigcup_{c,d} DT(n^c)|_{n^d}$

Proof. "$\subseteq$". Let $L \in \mathbb{P}/\text{poly}$ then by definition exists polynomial time circuit $\{C_n\}$ that recognizes L. We set $\{\alpha(n)\} = \{C_n\}$ (via encoding) and the DTM M simulates C_n on x. As circuit is of polynomial size, M works in polynomial time. Therefore

$$L \in \bigcup_{c,d} DT(n^c)|_{n^d}$$

"$\supseteq$". Let $L \in \bigcup_{c,d} DT(n^c)|_{n^d} \Rightarrow \exists DTMM$ which works in time $T(n)$, uses advice $\{\alpha(n)\}$ of size $a(n)$ and $L = M(x, \alpha_n)$.
Like in proof $\mathbb{P} \subseteq \mathbb{P}/\text{poly}$ 5.16 there exists a polynomial size family of $\{C_n\}$:

$$\forall x \in \{0,1\}^n \forall \alpha \in \{0,1\}^{a(n)} : M(x,\alpha) = C_m(x,\alpha), m = n + a(n)$$

Now we hardcode $\{\alpha_n\}$ into the circuit. Define

$$C_m^{hard}(x) = C_m(x, \alpha_n)$$

Then

$$x \in L \iff M(x,\alpha_n) = 1 \iff C_m(x,\alpha_n) = 1 \iff C_m^{hard}(x) = 1 \Rightarrow L \in \mathbb{P}/\text{poly}$$

□

Note 6.4. Not every boolean function can be computed by a polynomial size circuit.

Theorem 6.5 ($\exists f$ no poly circuit).

$$\forall n > 1, \exists f : \{0,1\}^n \to \{0,1\}$$

such that f cannot be computed by circuit $|C| = \frac{2^n}{10n}$.

Proof. Idea: show the mismatch between # of functions and # of circuits.
Let C be a circuit, $|C| = m$. Since every elementary gate has at most 2 inputs, C has $\leq 2m$ edges. Therefore C can be represented by a binary string of length $3m \log m$. So

$$|\{C| \ |C| = m\}| = 2^{3m \log m}$$

However, there are 2^{2^n} functions.
Setting $m = \frac{2^n}{10n}$ we have at most

$$2^{3\frac{2^n}{10n} \log(\frac{2^n}{10n})} \leq 2^{3\frac{2^n}{10n} n} = 2^{\frac{3}{10}2^n} < 2^{2^n}$$

circuits. □

Consequence 6.6. *With growing n more and more functions are not computable by polynomial circuits.*

Note 6.7 (Open Problem). If we can find a family of functions $\{f_n\}$ that are not computable by polynomial size circuit such that $L \in \mathbb{NP}$. Would imply $\mathbb{NP} \not\subseteq \mathbb{P}/\text{poly} \Rightarrow \mathbb{P} \neq \mathbb{NP}$.

Definition 6.8 (Π_n, Σ_n-SAT).

$$\Sigma_n - SAT = \exists x_1 \forall x_2 \ldots Q x_n \varphi(x_1, \ldots, x_n) = 1$$

$$\Pi_n - SAT = \forall x_1 \exists x_2 \ldots Q x_n \varphi(x_1, \ldots, x_n) = 1$$

Where Q is either $\exists$ of $\forall$ depending on parity of n, $\varphi \in CNF$.

Lemma 6.9 (Π_n, Σ_n-complete language). *$\Sigma_n - SAT$ is Σ_n-complete. $\Pi_n - SAT$ is Π_n-complete.*

Proof. For Σ_n only as proof for Π_n is the same.
Let $L \in \Sigma_n$ arbitrary. By the PH definition via alternating quantifiers definition 3.22:

$$\exists p \exists L_{po} \in \mathbb{P} : x \in L \iff \exists^{p(n)} y_1, \forall^{p(n)} \dots Q^{p(n)} y_n : (x, y_1, \dots, y_n) \in L_{po}$$

Let M be a DTM which accepts L_{po}. From Cook-Levin theorem we can encode the computation of M by a CNF φ_n:

$$x \in L \iff \exists^{q(|x|)} t_1, \forall^{q(|x|)} \dots Q^{q(|x|)} t_n : \varphi_n(v, t_1, \dots, t_n) = 1$$

where v encodes input x and t_i encodes y_i (both as binary string). All encodings are of size $q(|x|)$.
L can be a language over any alphabet, however the blowup of binary encoding would be logarithmic with respect to alphabet size. □

Theorem 6.10 (Karp-Lipton(1980)). $\mathbb{NP} \subseteq \mathbb{P}/\text{poly} \Rightarrow PH = \Sigma_2^p$.

Proof. Suffices to prove $\Pi_2 \subseteq \Sigma_2$ rest follows from 3.25??? To do that we need to prove

$$\exists L \in \Pi_2\text{-complete } \& L \in \Sigma_2$$

Idea: show Π_2-SAT $\in \Sigma_2$.
Π_2-SAT is a language of following CNF formulas:

$$\forall x \in \{0,1\}^n \exists y \in \{0,1\}^n : \varphi(x, y) = 1 \qquad (*)$$

If we fix x we get a language

$$L_x = \{(\varphi, x) |\ \exists y \in \{0,1\}^n : \varphi(x, y) = 1\}$$

Then

$$\forall x : L_x \in \Sigma_1 = \mathbb{NP} \overset{\text{assumption}}{\subseteq} \mathbb{P}/\text{poly}$$

Therefore $\exists \{C_k\} : |C_k| = p(k)$ such that

$$\forall \varphi_{n+m} \forall x \in \{0,1\}^n : C_k(\varphi_{n+m}, x) = 1 \iff \forall y \in \{0,1\}^n \varphi(x, y) = 1$$

Where k is # of bits to encode (φ, x).
Main idea: change $\{C_k\}$ to a new family $\{C_k^{out}\}$ which also outputs the satisfiable assignment. Meaning if C_k outputs 1 then C_k^{out} outputs $y \in \{0,1\}^n : \varphi(x, y) = 1$.
C_k^{out} outputs such y bit by bit by chaining circuits that do $\varphi \wedge y[i]$ where i is the bit position in y. We assume $|C_k^{out}| \leq q(n)$. We can encode C_k^{out} into a bit string of length $r(k) \leq \mathcal{O}(q^2(n))$.
Then,

$$\exists y \in \{0,1\}^{r(k)} \forall x \in \{0,1\}^n : \varphi(x, y) = C_k^{out}(\varphi, x)) = 1 \qquad (**)$$

Claim $(*) \iff (**)$ since if $(*)$ is true than $\exists$ circuit that on input (φ, x) outputs y such that $\varphi(x, y) = 1$. If $(*)$ is false $\Rightarrow \exists x : \forall y : \varphi(x, y) = 0$ then $(**)$ is also false.
Also $(**)$ is a Σ_2 formula, therefore Π_2-SAT $\in \Sigma_2$.

□

6.1 Classes AC, NC

Definition 6.11 (Class NC). $L \in NC^i$ if there exists polynomial size family of circuits $\{C_n\}_{n\in\mathbb{N}}$ accepting L such that $\forall n$ depth of $C_n \in \mathcal{O}(\log^i n)$.

$$NC = \bigcup NC^i$$

Definition 6.12 (Class AC). AC^i is defined similarly to NC^i but gates may have arbitrary # of input and output edges.

Theorem 6.13 (NC and AC).

$$\forall i : NC^i \subseteq AC^i \subseteq NC^{i+1}$$

Proof. By definition:

$$\forall i : NC^i \subseteq AC^i$$

Let $L \in AC^i$ arbitrary, then there is a circuit $\{C_n\}_{n\in\mathbb{N}}$ with depth $\mathcal{O}(\log^i n)$ that recognizes L. $\{C_n\}_{n\in\mathbb{N}}$ is a polynomial size family of circuits $\Rightarrow \exists p : |C_n| \leq p(n)$. Then, every gate in C_n has fan-in $\leq p(n)$. Replace every gate in C_n by a binary tree of fanin-2 gates with depth $\log p(n) \in \mathcal{O}(\log n)$. Therefore, we can construct C_n^{new} with fanin-2 of depth $\mathcal{O}(\log^{i+1} n)$ that simulates $C_n \Rightarrow L \in NC^{i+1}$. □

6.2 Connection between NC and parallel computing

Definition 6.14 (Parallel computer). Has n processors, communication between them with global clock. Maximum "distance" (shortest path in graph of processors) between processors is $\mathcal{O}(\log n)$.
For example hypercube architecture: labels of processors are binary numbers with $\log n$ bits. Processors are connected if their labels differ by 1 bit, Hamming distance is 1.

Definition 6.15 (Efficient parallel algorithm). A computational task is said to have *efficient parallel algorithms* if inputs of size n can be solved using a parallel computer with $n^{\mathcal{O}(1)}$ processors and in time $\log^{\mathcal{O}(1)} n$.
Source [1, p. 109].

Example 6.16 (Transitive closure). Problem of finding a transitive closure of directed graph can be solved by efficient parallel algorithm. Transitive closure adds an (a, b) edge for every $Path(a, b)$.
Instead of running multiple instances of BFS in parallel, we can multiple the adjacency matrix of a graph. The reachability matrix can be constructed in 1 multiplication $\mathcal{O}(1)$ using n^3 processors in time $\mathcal{O}(\log n)$.

Q: Do we assume that every processor outputs 1 bit?
A: No, each processor is a TM with output tape. As each of them works in log time, is uses at most log space which yields a polynomial size output.

Theorem 6.17 (NC and parallel computing). *L has an efficient parallel algorithm (which recognizes L) $\iff$ $L \in NC$.*

Proof. "⇒". Let efficient parallel algorithm for L has n^c processors and works in time $\mathcal{O}(\log^d n)$. Design a circuit $\{C_n\}$ with $\mathcal{O}(n^c \cdot \log^d n)$ processors.

"Gates" we construct (more precicely, fixed size circuits) are organized in $\mathcal{O}(\log^d n)$ levels. Gate at j-th level simulates processor in j-th step of the computation. The input edges comes from the previous levels and corresponds to the communication channels between the processors.
Then $L \in AC^d$, by 6.13 $L \in NC^d \Rightarrow L \in NC$.
"$\Leftarrow$" we have a polynomial size circuit with log depth. Replace every gate by a processor and interconnect them in some standard way. Sumilating C_n getting all signals from level j to $(j+1)$ may take $\mathcal{O}(\log n)$ steps of the parallel computing. Therefore simulation takes $\mathcal{O}(\log^{d+1} n)$. □

Q: what is the relationship between NC and $\mathbb{P}/\text{poly}$?
A: $NC, AC \subseteq \mathbb{P}/\text{poly}$, these classes allow fast computaton. Similar to linear or quadratic subclasses in $\mathbb{P}$.

7 Randomized computation

Definition 7.1 (Probabilistic TM (PTM)). TM with 2 transition functions: δ_1, δ_2. When PTM works on x it at each step selects δ_1 with probability $\frac{1}{2}$ (same for δ_2). PTM uses a fair cons toss that is independent from the previous results.
PTM M returns a random variable denoted $M(x)$, $M(x) = 1$ if M accepts x and 0 o/w. We say that M runs in tim $T(n)$ if for every input x M stops after at most $T(|x|)$ steps, regardless of the selections.

Notation 7.2. For a language $L \in \{0,1\}^*$ and $x \in \{0,1\}^*$ we define:

$$L(x) = 1 \iff X \in L \& L(x) = 0 \iff x \notin L$$

Definition 7.3 (BPP). PTM M accepts the language $L \in \{0,1\}^*$ in time $T(n)$ if

$$\forall x \in \{0,1\}^* : Pr[M(x) = L(x)] \geq \frac{2}{3}$$

And M runs in $T(n)$.
$BPTIME(T(n))$ is a class of languages accepted by PTM in time $T(n)$.

$$BPP = \bigcup BPTIME(n^c)$$

BPP - bounded error probabilistic polynomial time.

Properties 7.4 (BPP)**.**

BPP is invariant to (robust):

- Accepting probability: BPP will remain the same if we replace $\frac{2}{3}$ by any fraction $\in (\frac{1}{2}, 1)$.
- Probabilities of picking δ_1 or δ_2 can also be changed to $p, 1-p$ instead of $\frac{1}{2}$.
- Running time: $T(n)$ can be replaced by "expected" $T(n)$.

Note 7.5 (Open Question). We know:

$$\mathbb{P} \subseteq BPP$$

as we can take $\delta_1 = \delta_2$.
However the strict inclusion is an open question

$$\mathbb{P} \subsetneq BPP$$

Note 7.6 (BPP and NP). The idea is similar, but BPP requires $\frac{2}{3}$ of accepting computation branches and $\mathbb{NP}$ only 1.

Definition 7.7 (Probabilistic TM (PTM) Alternative). Randomness can be presented making PTM accept additional input string, which is random $y \in \{0,1\}^{T(n)}$. Each bit tells which transition function to choose.

Theorem 7.8 (Chernoff bound). *Let $x_i, \ldots, x_n$ be independent r.v. such that $x_i \in \{0,1\}$ and let $\mu = \sum \mathbb{E}[x_i]$ then*

$$\forall c > 0 : \Pr[|\sum x_i - \mu| \geq c \cdot \mu] \leq 2e^{-\min\{\frac{c^2}{4}, \frac{c}{2}\} \cdot \mu}$$

Theorem 7.9 (BPP error reduction). *Let $L \in \{0,1\}^*$ be a language, $c > 0$ constant. Assume that there $\exists PTM\ M$ such that*

$$\forall x \in \{0,1\}^* : Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$$

Note that as x grows, error $|x|^{-c} \to 0$.
Then

$$\forall d > 0 : \exists PTM\ M_e : \forall x \in \{0,1\}^* : Pr[M_e(x) = L(x)] \geq 1 - 2^{-|x|^d}$$

And if M work in polynomial time so does M_e.

Proof. Idea: M_e runs M $k = 8 \cdot |x|^{2d+c} + 1$ times. Assume k is odd. Obtaining results $y_1, y_2, \ldots y_k$ and M_e accepts iff

$$\sum y_i > \frac{1}{2}k$$

Majority decision.
Denote $x_i \in \{0,1\}^*$ r.v. where

$$x_i = 1 \iff y_i = L(x)$$

As each y_i represents different branch of computation, and coin flip is independent, x_i are independent. Then by assumption on M

$$\mathbb{E}[x_i] = \Pr[x_i = 1] \geq \frac{1}{2} + |x|^{-c} = p$$

Define $\delta = \frac{1}{2}|x|^{-c}$ and consider

$$S = \sum x_i \geq pk - \delta pk = pk(1-\delta) = k\left(\frac{1}{2}|x|^{-c}\right)\left(1 - \frac{1}{2}|x|^{-c}\right) = \frac{1}{2}k + k\left(\frac{3}{4}|x|^{-c} - \frac{1}{2}|x|^{-2c}\right)$$

Then as $\left(\frac{3}{4}|x|^{-c}-\frac{1}{2}|x|^{-2c}\right)\geq 0$

$$S=\frac{1}{2}k+k\left(\frac{3}{4}|x|^{-c}-\frac{1}{2}|x|^{-2c}\right)>\frac{1}{2}k$$

We conclude that if M answers correctly at least $>\frac{1}{2}k$ then M_e answers correctly (for sure).
On the contrary, M_e can make a mistake only if

$$\sum x_i < pk-\delta pk$$

So

$$\Pr[\sum x_i < pk-\delta pk]$$

is an upper bound of the probability that M_e makes a mistake. Then

$$\mathbb{E}[x_i]\geq p \Rightarrow \sum \mathbb{E}[x_i]\geq pk$$

For some $\varepsilon>0$ denote

$$\mu=\sum \mathbb{E}[x_i]=pk+\varepsilon$$

Finally compute

$$P_e=\Pr[pk-\sum x_i>\delta pk]\leq \Pr[pk-\sum x_i+(1-\delta)\varepsilon>\delta pk]=\Pr[pk+\varepsilon-\sum x_i>\delta pk+\delta\varepsilon]=$$

$$=\Pr[\mu-\sum x_i>\delta\mu]=$$

As the probability is symmetric and by Chernoff 7.8

$$=\frac{1}{2}\Pr[|\mu-\sum x_i|>\delta\mu]\leq \frac{1}{2}2e^{-\frac{\delta^2}{4}\mu}\leq e^{-\frac{\delta^2}{4}pk}$$

To summarize:

$$P_e=\Pr[M_e \text{ makes an error}]\leq e^{-\frac{1}{4}\frac{|x|^{-2c}}{4}(\frac{1}{2}+|x|^{-c})8\cdot|x|^{2d+c}+1}=e^{-\frac{8\cdot|x|^{2c+d}}{32\cdot|x|^{2c}}+\frac{8\cdot|x|^{2c+d}}{16\cdot|x|^{3c}}}$$

By reducing the exponent (2nd summand) and simplifying the first

$$P_e\leq e^{-\frac{1}{4}|x|^d}=(e^{\frac{1}{4}})^{-|x|^d}\leq 2^{-|x|^d}$$

□

Lemma 7.10 (Fair coin). *A fair coin can be simulated by a PTM with an access to a biased coin with* $\Pr[head]=p\in(0,1)$ *in* $\mathcal{O}(1)$ *expected time.*

Proof. In each round PTM flips the biased coin twice if

$$\begin{cases} H+H & \text{for} \\ T+T & \text{for} \end{cases} \Rightarrow \text{new pair of flips}$$

Otherwise

$$\begin{cases} H+T & \text{for } H, \Pr[H]=p(1-p) \\ T+H & \text{for } T, \Pr[H]=(1-p)p \end{cases}$$

As

$$\Pr[\text{PTM stops at round } i]=2p(1-p)=c, c\in(0,1)$$

Then expected # of trials before PTM stops

$$\sum i(1-c)^{i-1}c$$

which converges. □

Lemma 7.11 (Biased coin (Von-Neumann)). *A biased coin with probability* $\Pr[head] = \rho$ *can be simulated in* $\mathcal{O}(1)$ *time by a "standard" PTM (with fair coin) if the* i*-th bit of* ρ *can be generated in poly*$(n) = n^c$.

Proof. Let ρ be a binary written as

$$\rho = 0p_1p_2\dots$$

PTM generates fair sequence $b_1, b_2, \dots$. If PTM in i-th step generates b_i then

- $b_i < p_i \Rightarrow H$
- $b_i > p_i \Rightarrow T$
- $b_i = p_i$ go to step $(i+1)$.

M gets to i-th step $\iff$

$$\forall 1 \leq j \leq i-1 : b_j = p_j$$

which occurs with probability $\frac{1}{2^{i-1}}$. $p_i = 1 \Rightarrow$ return H in step i with probability $\frac{1}{2}$.
$p_i = 0 \Rightarrow$ return H in step i with probability 0. Then

$$\Pr[H] = \sum p_i \frac{1}{2^i} = \rho$$

The expected time $\sum n^i \frac{1}{2^{i-1}}$ which converges. □

Definition 7.12 (Expected running time). Let M be a PTM and x an input. Then $T_{M,x}$ is a random variable which denotes the time it takes M to halt on x. The PTM M runs in expected time $T(n)$ if $\forall x \in \{0,1\}^* : \mathbb{E}[T_{N,x}] \leq T(|x|)$.

Theorem 7.13 (Markov inequality). *Let* X *be a non-negative r.v. Then*

$$\Pr[X \geq k\mathbb{E}[x]] \leq \frac{1}{k}$$

Observation 7.14. Let PTM M have expected time complexity $T(n)$. If we simulate M on PTM M_s with time complexity $100T(n)$ (by rejecting longer computations), then the probability (by Markov 7.13)

$$\Pr[M \text{ runs longer than } 100T(n)] \leq \frac{1}{100}$$

Therefore, M_s rejects if it stops because of the clock, o/w it does the same as M. Then $\Pr[M_s$ makes an error$]$ is at most 100 greater than for M.
We can reduce the error back to the error of M by 7.9.

7.1 Classes RP, ZPP

Definition 7.15 (RTIME, RP). $L \in \text{RTIME}(T(n)) \Leftrightarrow \exists$ PTM M working in (expected) time $T(n)$ such that

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq 2/3$$
$$x \notin L \Rightarrow \Pr[M(x) = 0] = 1$$

$RP = \bigcup_c \text{RTIME}(n^c)$ – Randomized Polynomial time
Note that we can again prove that class is the same with either expected or exact polynomial time.

Definition 7.16 (ZTIME, ZPP). $L \in \text{ZTIME}(T(n)) \Leftrightarrow \exists$ PTM M working in (expected) time $T(n)$ such that

$$x \in L \Rightarrow \Pr[M(x) = 1] = 1$$
$$x \notin L \Rightarrow \Pr[M(x) = 0] = 1$$

$ZPP = \bigcup_c \text{ZTIME}(n^c)$ – Zero-error Probabilistic Polynomial time
Note that we can again prove that class is the same with either expected or exact polynomial time.

Lemma 7.17 (On probabilistic classes). *1.* $RP \subseteq \mathbb{NP}$

2. $BPP = co - BPP$

3. $RP \subseteq BPP, co - RP \subseteq BPP$

4. $ZPP = RP \cap co - RP$

Proof. 1. For $\mathbb{NP}$ we require at least one accepting computation, by definition of RP we have at least $\frac{2}{3}$ of them. If we allow PTM to work in expected polynomial time, then there is shorter accepting branch of computation.

2. Design new PTM which simulates original PTM but flips all answers.

$$\Pr[\overline{M}(x) = \overline{L}(x)] = \Pr[1 - M(x) = 1 - L(x)] = \Pr[M(x) = L(x)] \geq \frac{2}{3}$$

3. Follows from stricter definition of RP. Second condition follows from 2).

4. $ZPP \subseteq RP \cap co - RP$ Follows from the stricter definition of ZPP.

   $ZPP \supseteq RP \cap co - RP$. Let $L \in RP \cap co - RP$ arbitrary. By definition, we have 2 PTM: M_y, M_n.

$$x \in L \Rightarrow \Pr[M_y(x) = 1] \geq 2/3 \& \Pr[M_n(x) = 1] = 1$$
$$x \notin L \Rightarrow \Pr[M_y(x) = 0] = 1 \& \Pr[M_n(x) = 1] \geq 2/3$$

   Then, if $M_y(x) = 1 \Rightarrow x \in L$ (for sure) and symmetrically $M_n(x) = 1 \Rightarrow x \notin L$ (for sure). Construct new PTM M that simulates both PTM in parallel, output depending on the simulation:

   - $M_y(x) = 1 \Rightarrow M(X) = 1$
   - $M_n(x) = 0 \Rightarrow M(X) = 0$
   - $M_y(x) = 0 \& M_n(x) = 1 \Rightarrow$ try simulation again

   Then

$$x \in L \Rightarrow \Pr[M_y(x) = 0 \& M_n(x) = 1] \leq \frac{1}{3} \cdot 1 = \frac{1}{3}$$
$$x \notin L \Rightarrow \Pr[M_y(x) = 0 \& M_n(x) = 1] \leq 1 \cdot \frac{1}{3} = \frac{1}{3}$$

   Expected # of repetitions of the simulation is $\leq \sum i \cdot (\frac{1}{3})^i$. Which converges.

□

Theorem 7.18 ($BPP \subseteq \mathbb{P}/\text{poly}$ (Adleman)).

$$BPP \subseteq \mathbb{P}/\text{poly}$$

Proof. $L \in BPP \Rightarrow \exists DTM\ M$ with 2 inputs $x \in \{0,1\}^n, r \in \{0,1\}^m, m = poly(n)$ such that (by 7.9):

$$\Pr[M(x,r) \neq L(x)] \leq \frac{1}{2^{n^2}} \leq \frac{1}{2^{m+1}}$$

$r \in \{0,1\}^m$ is *bad* for $x \in \{0,1\}^m$ if $M(x,r) \neq L(x)$. For a fixed x there are $\frac{2^m}{2^{n+1}}$ strings r that are bad for x. For all x there exists at $\leq 2^n \cdot \frac{2^m}{2^{n+1}} = 2^m/2$ bad strings. Therefore $\geq \frac{1}{2}2^m$ strings are good $\forall x$.
Pick some r_0 that is good for x. From $\mathbb{P} \subseteq \mathbb{P}/\text{poly}$ 5.16 M can be simulated by polynomial size family of circuits $\{C_{m+n}\}$. Note that value of r_0 is hard wired into the circuit. Then

$$\forall x \in \{0,1\}^n : L(x) = M(x,r_0) = C_{m+n}(x) \Rightarrow L \in \mathbb{P}/\text{poly}$$

□

Theorem 7.19 (Sipser-Gács-Lautemann). $BPP \subseteq \Sigma_2^p \cap \Pi_2^p$

Proof. Since BPP is closed under complements lemma 7.17 it is enough to show

$$BPP \subseteq \Sigma_2^p$$

$L \in BPP \Rightarrow \exists DTM\ M$ with 2 inputs, size of random bits is m, such that (by 7.9 setting $d = 1$):

$$x \in L \Rightarrow \Pr[M(x,r) = 1] \geq 1 - \frac{1}{2^n}$$
$$x \notin L \Rightarrow \Pr[M(x,r) = 1] \leq \frac{1}{2^n}$$

Define

$$\forall x \in \{0,1\}^n : S_x := \{r |\ M(x,r) = 1\}$$

Trivially

$$x \in L \Rightarrow |S_x| \geq (1 - \frac{1}{2^n}) \cdot 2^m$$
$$x \notin L \Rightarrow |S_x| \leq \frac{1}{2^n} \cdot 2^m$$

Also define

$$\forall k,l \in \{0,1\}^m : k + l = k\ XOR\ l = k + l \mod 2$$
$$\forall Z \subseteq \{0,1\}^m, u \in \{0,1\}^m : Z + u = \{z + u |\ z \in Z\}$$

Lemma 7.20 (Sipser-Gács-Lautemann 1). *Let $Z \subseteq \{0,1\}^m \& |Z| \leq 2^{m-n}$. Also let $u_1, \ldots u_k \in \{0,1\}^m$. Then for $k = \lceil \frac{m}{n} + 1 \rceil$.*

$$\bigcup (Z + u_i) \subsetneq \{0,1\}^m$$

Proof. As in Lagrange theorem for groups

$$\forall i : |Z + u_i| = |Z|$$

Therefore, even if all $Z + u_i$ are disjoint:

$$\bigcup (Z + u_i) \leq k|Z| \leq \lceil \frac{m}{n} + 1 \rceil \cdot \frac{2^m}{2^n} \leq 2^m$$

Holds for large enough n as $m = poly(n)$ but the denominator is exponential 2^n:

$$\frac{m}{n2^n} < 1$$

□

Lemma 7.21 (Sipser-Gács-Lautemann 2). *Let* $Z \subseteq \{0,1\}^m \& |Z| \geq (1 - 2^m)2^n$. *Then*

$$\exists u_1, \ldots u_k \in \{0,1\}^m : \bigcup (Z + u_i) = \{0,1\}^m$$

Proof. It is enough to prove that for a random choice of $u_1, \ldots u_k$ we get

$$\Pr[\bigcup (Z + u_i) = \{0,1\}^m] > 0 \iff \Pr[\bigcup (Z + u_i) \subsetneq \{0,1\}^m] < 1$$

Which is equivalent

$$\Pr[\exists r \in \{0,1\}^m : r \notin \bigcup (Z + u_i)] < 1$$

Define r is *bad* for $i \iff r \notin Z + u_i \overset{XOR}{\iff} r + u_i \notin Z$. Pick $u_1, \ldots u_k \in \{0,1\}^m$ uniformly independently. Then // TODO fix (n, m)

$$\forall r : \Pr[r + u_i \in Z] \geq 1 - 2^m \Rightarrow \Pr[r \text{ is bad for } i] \leq 2^{-n}$$

By independent choice of u_i:

$$\Pr[r \text{ is bad } \ \forall i] < 2^{-nk}$$

$$\Pr[\exists r \text{ is bad } \ \forall i] = \Pr[\exists r : r \notin \bigcup (Z + u_i)] < 2^m \cdot 2^{-nk} \overset{nk \geq m}{<} 2^m \cdot 2^{-m} = 1$$

□

$$x \in L \iff \exists u_1, \ldots, u_k \in \{0,1\}^m : \forall r \in \{0,1\}^m : r \in \bigcup (Z + u_i)$$

Equivalently

$$x \in L \iff \exists u_1, \ldots, u_k \in \{0,1\}^m : \forall r \in \{0,1\}^m : \vee_i^k M(x, r + u_i) = 1$$

As M is deterministic and works in polynomial time and we use it $k = \lceil \frac{m}{n} + 1 \rceil$ times: $\vee_i^k M(x, r + u_i)$ can be checked in polynomial time.

$$\vee_i^k M(x, r + u_i) = 1 \in \mathbb{P}$$

The formula is

$$\exists \forall \mathbb{P} \in \Sigma_2$$

□

8 Ladner theorem

Theorem 8.1 (Ladner). *If* $\mathbb{P} \neq \mathbb{NP} \Rightarrow \exists L \in \mathbb{NP} \setminus \mathbb{P} \& L \notin \mathbb{NP}$*-complete.*
Proves existence of $\mathbb{NP}$*-intermediate problems. Candidates for the* $\mathbb{NP}$*-intermediate problems are* Factoring *or* Graph isomorphism*.*

Proof. Idea: find a language $L \in \mathbb{NP} \cap co-\mathbb{NP} \wedge L \notin P$. Language we construct is artificial. Not all technical details of the proof will be presented.
Redefine SAT as

$$SAT = \{i \in \{0,1\}^* \mid \text{CNF formula with code i is satisfiable}\} \in \mathbb{NP}$$

Let $M_1, M_2, \ldots$ be a list of oracle DTM using $\{0,1\}$ alphabet and running in polynomial time.
Note:

$$x \in \mathbb{P} \Rightarrow \exists i : X = L(M_i, \emptyset)$$
$$x \in \mathbb{NP}-c \Rightarrow \exists i : SAT = L(M_i, X)$$

Our goal is to construct language A such that:

$$A \notin \{L(M_1, \emptyset), L(M_2, \emptyset), \ldots\}$$
$$SAT \notin \{L(M_1, A), L(M_2, A), \ldots\}$$

A is defined as $L(N)$ where N is an universal NTM that works in time $p(n)$ and uses subroutines:

Agree($M_i, \emptyset, y$**)**

1. N simulates M_i with oracle $\emptyset$ on y and outputs YES if M_i accepts. Outputs NO if M_i rejects.
2. N checks if $y \in A$. //here A simulates itself
3. if 1) and 2) have the same result - output YES, o/w NO.

Agree-O(M_i, A, y**)**

1. N simulates M_i with oracle A on y and outputs YES if M_i accepts. Outputs NO if M_i rejects.

   //here A simulates itself
2. N checks if $y \in SAT$.
3. if 1) and 2) have the same result - output YES, o/w NO.

N works on input $x \in \{0,1\}$ with following algorithm (i is the Godel number of TM, y is current string)

1. $i = 1, y =$ "" // empty string

2. until(N makes $p(|x|)$ steps) repeat
3. while(AGREE$(M_i, \emptyset, y)$) do $y = \text{next}(y)$ // increment
4. while(AGREE-O(M_i, A, y)) do $y = \text{next}(y)$ // increment
5. $++i$ // get next TM
6. if main loop terminates in step 3 then N accepts $\iff x \in SAT$.
7. else N rejects x.

$A \in \mathbb{NP}$ as it is accepted by NTM in polynomial time.

$A \notin \mathbb{NP}$-complete Assume by contradiction that $A \in \mathbb{NP}$-complete. Then $\exists k : SAT = L(M_k, A)$ also let k be smallest possible. Let $j < k$ be arbitrary and let x be large enough. As by assumption $A \in \mathbb{NP}$-complete $\Rightarrow A \notin \mathbb{P}$ as it would contradict $\mathbb{P} \neq \mathbb{NP}$. Therefore $A \neq L(M_j, \emptyset) \Rightarrow \exists y : AGREE(M_j, \emptyset, y)$ outputs NO. Then main loop *breaks* from "while" in step 3, because N will eventually find such y.
Moreover, main loop *breaks* from "while" in step 4 for some $l \neq k$ as $SAT \neq L(M_l, A) \Rightarrow \exists y : AGREE - O(M_l, A, y)$ outputs NO. Finally, main loop never leaves step 4 for k.
Altogether, $\exists c, \forall x : |x| \geq c$ the main loop terminates in step 4 and rejects x. So, N accepts a finite language $\Rightarrow A \in P$ which is a contradiction.

$A \notin \mathbb{P}$ Assume by contradiction that $A \in \mathbb{P}$. Then $\exists k : A = L(M_k, \emptyset)$ also let k be smallest possible. Let x be large enough.
For $j < k$ main loop *breaks* from "while" in step 3 as we assume k is the smallest. Also, main loop *breaks* from "while" in step 4 as we assume k is the smallest. As $A \in \mathbb{P} \& SAT \neq L(M_j, A)$. Finally, main loop never leaves step 3 for k. Therefore, N can accept x only in step 2 of Agree procedure. So

$$\exists c, \forall x : |x| \geq c : x \in A \iff x \in SAT \Rightarrow A \in NPC$$

Which is a contradiction with assumption $A \in \mathbb{P}$.

Technical details Note that hardest technical detail is the way N simulates A (itself). Q: again lier's paradox (self-reference)?
Also, N has to check 2 conditions in both subroutines. Therefore, if the branch is accepting, N returns to the branching point and compares the results. However, such approach does not work for rejecting branches. One of the solution is to choose $|y| \leq \log |x| \Rightarrow 2^{|y|} \leq |x|$ and N can check all branches. □

9 PCP theorem

Many problems in $\mathbb{NP}$ are artificial versions of optimization problems.

Example 9.1 (Knapsack). Having n items with sizes $s_1, \ldots, s_n$ and prices $p_1, \ldots, p_n$. Also capacity c - size of knapsack.
Optimization problem: minimize $\sum p_i$ subject to constraint $\sum s_i \leq c$. Optimization knapsack is $\mathbb{NP}$-hard.

Decision problem: add a parameter k. Input: $(s_1, \ldots, s_n, p_1, \ldots, p_n, c, k)$.
Question: $\exists A \subseteq [n] : \sum_{i \in A} p_i \geq k \& \sum_{i \in A} s_i \leq c$.
Decision knapsack is $\mathbb{NP}$-complete.

Example 9.2 (Subset sum). Is a modification of Knapsack.
Input: $a_1, \ldots, a_n, b$.
Question: $\exists A \subseteq [n] : \sum_{i \in A} a_i = b$.
Reduction from Knapsack: set $s_i = a_i \& c_i = a_i$.

Despite Knapsack being $\mathbb{NP}$-hard, there is an approximation algorithm.

Definition 9.3 (FPTAS). Fully polynomial time approximation scheme. Algorithm A with 2 inputs:

1. size of the problem x.
2. allowed precision (relative error) ε.

Such that $\frac{|A(x) - OPT(x)|}{OPT(x)} \leq \varepsilon$. A runs in polynomial time with respect to $|x|$ and $\frac{1}{\varepsilon}$.

Knapsack has a FPTAS algorithm, therefore it can be efficiently approximated with arbitrary error. On the other hand, Traveling salesman problem (TSP) does not have such algorithm.

Example 9.4 (TSP). Input: complete undirected graph $G = (V, E)$ and function $l : E \to \mathbb{N}$.
Output: shortest Hamiltonian circuit in G.

Lemma 9.5 (Inaproximation of TSP). *If there exists polynomial time approximation algorithm A for TSP with ratial error $\rho = \frac{A(x)}{OPT(x)}$. Where $\rho \geq 1$ is constant. Then $\mathbb{P} = \mathbb{NP}$.*

Proof. Let us take an instance of HC problem: graph $G = (V, E), |V| = n$, question: $\exists HC \in G$? And convert it into TSP problem

$$G_t = (V, E = V \times V)$$

and the lengths:

$$l(e) = \begin{cases} 1 & \text{for } e \in E \\ \rho n + 1 & \text{for } e \notin E \text{ artificial edge} \end{cases}$$

If the initial problem is answered YES then $OPT(x) = n$ and the algorithm A for TSP outputs a HC of length $\leq n\rho$. Therefore, A must output HC which was in the initial graph.
On the contrary, if the initial answer is NO, then A outputs $OPT(x) > n\rho$. GAP construction no HC in $(n, n\rho]$.
Therefore a polynomial approximation algorithm for TSP gives a polynomial algorithm for HC. □

Definition 9.6 (MAX-3-SAT). For a 3-CNF formula $\varphi, val(\varphi) =$ max fraction of clauses that can be satisfied by an assignment.
Note: φ is satisfiable $\iff val(\varphi) = 1$.
For $\rho \leq 1$ and algorithm A is ρ-approximation algorithm in MAX-3-SAT if $\forall \varphi$ with m clauses $A(\varphi)$ outputs and assignment satisfying $\geq \rho \cdot val(\varphi) \cdot m$ clauses.
In other words $OPT(\varphi) = n$ and

$$\frac{A(\varphi)}{OPT(\varphi)} \geq \rho \cdot m$$

Exercise 9.7 (MAX-3-SAT algorithm). Greedy algorithm for MAX-3-SAT with $\rho = \frac{1}{2}$:

1. take variable x_i
2. count all clauses where x_i or $\neg x_i$ is present. Take majority and assign $x_i = 1, x_i = 0$ (resp) and remove opposite.

At every step we satisfy at least half of clauses that contains x_i.
What is the expected # of satisfied clauses for a random assignment? Assume, every clause has 3 distinct variables, therefore is has 8 possible assignments and only 1 of them does not satisfy the clause. Therefore, $\mathbb{E}[rng] \geq \frac{7}{8} \cdot m$ yielding $\rho = \frac{7}{8}$. Also, $\forall \varepsilon > 0$ there is a deterministic algorithm with $\rho = \frac{7}{8} - \varepsilon$ (derandomization).

Note 9.8. PCP theorem implies that if there is an approximation algorithm with $\rho = \frac{7}{8} + \varepsilon \Rightarrow \mathbb{P} = \mathbb{NP}$.

Exercise 9.9 (Vertex cover algorithm). Input: undirected graph $G = (V, E)$
Output: minimal vertex cover:

$$S \subseteq V : \forall e = (a, b) \in E : \{a, b\} \cap S \neq \emptyset$$

Even though, Vertex cover is the dual problem for Independent set, the approximation approaches are different.

Reminder 9.10 ($\mathbb{NP}$ verifier). Verifier for $L \in \mathbb{NP}$ is a polynomial time DTM:

$$x \in L \Rightarrow \exists y : V^y(x) = 1$$
$$x \notin L \Rightarrow \forall y : V^y(x) = 0$$

and $|y| = poly(|x|)$.

Definition 9.11 (Probabilistic verifier). Verifier has a *remote access* to the proof, where proof is some string $\in \{0,1\}^*$. Verifier can access any position independently in constant time.

Definition 9.12 (PCP verifier). Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has a $r(n), q(n)$ PCP verifier if there is a **polynomial time** probabilistic algorithm A which satisfies the following:

- Efficiency: on input string $x \in \{0,1\}^*$ given a random access to string $y \in \{0,1\}^*$. A uses at most $r(n)$ random bits and makes at most $q(n)$ queries to some locations in y. We denote by $A^y(x)$ a r.v. representing output of A on input x with access to proof y.
- Completeness: $x \in L \Rightarrow \exists y \in \{0,1\}^* : \Pr[A^y(x) = 1] = 1$
- Soundness: $x \notin L \Rightarrow \forall y \in \{0,1\}^* : \Pr[A^y(x) = 1] \leq \frac{1}{2}$

Note that the proof can be exponential, we only care about the polynomial size addressing.

Definition 9.13 (PCP class). $L \in PCP(r(n), q(n))$ if $\exists c, d > 0 : L$ asks a $(c \cdot r(n), d \cdot q(n))$-PCP verifier.

Lemma 9.14 (PCP properties). *1. We may assume for a (r, q)-verifier that every proof y*

$$|y| \leq q \cdot 2^{r(n)}$$

as the # of queries is bounded by q and $2^{r(n)}$ possible outcomes of random access.

2. *$PCP(r(n), q(n)) \subseteq NT(2^{\mathcal{O}(r(n))} \cdot q(n))$. We construct an NTM which guesses the proof of size $2^{\mathcal{O}(r(n))} \cdot q(n)$. M deterministically runs the verifies on all possible choices of the advice. M accepts $\iff$ the verifier accepts in ALL cases.*

Theorem 9.15 (PCP).

$$\mathbb{NP} = PCP(\log n, 1)$$

Proof. "$\subseteq$" follows from lemma 9.14.

$$PCP(\log n, 1) \subseteq NT(2^{\mathcal{O}(\log n)}) = \mathbb{NP}$$

"$\supseteq$" hard. □

Example 9.16 (PCP system). Let GNI be pairs of non-isomorphic graphs. Input: (G_n, G_i) graphs on n vertexes.
Claim: GNI $\in PCP(\log n, 1)$.
The proof will be a binary string with each location being index by adjacency matrix of a graph on n vertexes. Assume that for correct y:

$$y[H] = \begin{cases} 0 & \text{for } H \equiv G_0 \\ 1 & \text{for } H \equiv G_1 \\ \in \{0,1\} & \text{for } else \end{cases}$$

Verifies takes at random some $b \in \{0,1\}$ than randomly picks a permutation of $[n]$ which defines $H \equiv G_0$, guesses $y[H]$ and accepts $\iff y[H] = b$.
If $x \in GNI$ then bits corresponding to G_0 and G_1 are disjoint. With such proof the verifier should succeed, completeness is satisfied.
Otherwise, there are positions in y which corresponds to both G_0 and G_1. By the definition of $y[H]$, these bits are random, so the probability of success if $\leq \frac{1}{2}$.

Corollary 9.17. *$\exists \rho < 1$ such that if there is an polynomial size approximation algorithm for MAX-3-SAT then $\mathbb{P} = \mathbb{NP}$.*

Definition 9.18 (qCSP). A $qCSP$ instance φ is a collection $\varphi_1, \ldots, \varphi_n$ of functions (constraints): $\{0,1\}^n \to \{0,1\}$ such that each function φ_i depends on at most q variables. We say that an assignment $u \in \{0,1\}^n$ satisfies constraint φ_i if $\varphi_i(u) = 1$. The fraction of constraint satisfied by u is $\sum \frac{\varphi_i(n)}{n}$ and let $val(\varphi_i)$ max of this value over all $u \in \{0,1\}^n$. Also φ is satisfiable if $val(\varphi) = 1$.
3-SAT is a special case of qCSP, as each clause is equivalent to a function φ_i.

Definition 9.19 (Gap CSP). $\forall q \in \mathbb{N}, \rho \leq 1$ a ρ-GAP-qCSP is a problem determining whether a given qCSP instance has $val(\varphi) = 1$ or $val(\varphi) < \rho$.
We say that ρ-GAP-qCSP is $\mathbb{NP}$-hard if there is a polynomial time function f mapping strings to representations of qCSP satisfying

$$x \in L \Rightarrow val(f(x)) = 1$$
$$x \notin L \Rightarrow val(f(x)) < \rho$$

Theorem 9.20 (PCP 1). *There exists $\rho < 1 : \forall L \in \mathbb{NP}$ there is polynomial time function f mapping string to representations of 3CNF:*

$$x \in L \Rightarrow val(f(x)) = 1$$
$$x \notin L \Rightarrow val(f(x)) < \rho$$

As 3SAT is $\mathbb{NP}$-complete, we can always map instance of L to satisfiable or unsatisfiable 3CNF. Theorems 2nd claim is stronger: as the standard reduction only guarantees 1 unsatisfiable clause. In this case, we get constant fraction of unsatisfiable clauses.

Theorem 9.21 (PCP 1 stronger). $\forall \varepsilon > 0, \forall \rho = \frac{7}{8} + \varepsilon : \forall L \in \mathbb{NP}$ *there is polynomial time function f mapping string to representations of 3CNF:*

$$x \in L \Rightarrow val(f(x)) = 1$$
$$x \notin L \Rightarrow val(f(x)) < \rho$$

Theorem 9.22 (PCP 2). $\exists q \in \mathbb{N}, \rho < 1 : \rho$*-GAP-qCSP is $\mathbb{NP}$-hard.*

Theorem 9.23 (PCP equivalence). *PCP 9.15 is equivalent to PCP 2 9.22.*

Proof. "$\Rightarrow$". Assume $\mathbb{NP} = PCP(\log n, 1)$. We show that $\frac{1}{2}$-GAP-qCSP is $\mathbb{NP}$-hard by a reduction from 3-SAT to $\frac{1}{2}$-GAP-qCSP. By assumption, there is a PCP system with polynomial time verifier V which makes q queries (constant) and uses $c \cdot \log n$ random bits. Define $\forall x : V_{x,r}$ function whether V accepts x with random coins r. Separate collection for every input. Which depends on at most q variables, as we make at most q queries. q is a constant. Taking φ as collection of all functions they form a $qCSP$. Does it satisfy GAP-qCSP?
If we get an instance of 3-SAT which is satisfiable, then by completeness all constraints should be satisfied.
On other hand, with a non-satisfiable instance of 3-SAT by soundness at least half of the constraints will give 0. Then $val(\varphi) \leq \frac{1}{2}$.
Note that

- PCP verifier $V \iff CSP$ instance φ.
- PCP proof $y \iff$ assignment of variables in φ
- length of the proof $\iff$ # of variables. We know: # of variables $\leq q \cdot 2^r = q \cdot 2^{c \log n} = q \cdot n^c$.

"$\Leftarrow$". Assume ρ-GAP-qCSP is $\mathbb{NP}$-hard for some constant ρ, q. Take $L \in \mathbb{NP}$ arbitrary we want to design PCP system with parameters $(\log n, 1)$ for L. Let f be a polynomial time reduction from $L \to \rho$-GAP-qCSP The verifier on input x constructs $f(x) = \varphi$ which is a qCSP instance with m constraints. Verifier chooses at random $i \in [m]$ and checks φ_i by making q queries. The log parameter of CSP comes from guessing $\log m$ bits as an encoding of m.
If $x \in L \Rightarrow \varphi$ accepts with probability 1. On the other hand, $x \notin L \Rightarrow \varphi$ accepts with probability $< \rho$. We can adjust the probability to $\frac{1}{2}$ by running the verifier polynomial times. □

List of Theorems

List of Theorems

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.