CSC2401 Introduction to Computational Complexity

1 Computation Models and Time Complexity

What is Complexity Theory?

- How can computation problems be computed efficiently? How do we measure efficiency?
- How limited resources (time/space/randomness) affect computation?
- Complexity classes and relations between different classes.

Examples of problems: factoring, graph coloring, multiplying matrices, circuit satisfiability.

***Definition:* 1.1: Decision Problems**

For a language $L \subset \{0,1\}^*$. Given $x \in \{0,1\}^*$, decide if $x \in L$.

Example:

- Graph 3-coloring: Given a graph G, can we color the vertices of G with 3 colors s.t. adjacent vertices get different colors?
- Circuit SAT: Given a circuit, does it have a satisfying assignment?

***Definition:* 1.2: Search Problems**

Given $x \in \{0,1\}^*$, find $y \in \{0,1\}^*$ s.t. $x, y \in R \subset \{0,1\}^* \times \{0,1\}^*$.

Example:

- Graph 3-coloring: Given a graph G, find the coloring satisfying the constraint.
- Circuit SAT: Given a circuit, find a satisfying assignment.

***Definition:* 1.3: Counting Problems**

Given $x \in \{0,1\}^*$, find the number of $y \in \{0,1\}^*$ s.t. $x, y \in R \subset \{0,1\}^* \times \{0,1\}^*$.

Example:

- Graph 3-coloring: Given a graph G, find the number of satisfying colorings.
- Circuit SAT: Given a circuit, find the number of assignments.

1.1 Turing Machine

There are multiple computations models: Turing machines, circuits, interactive protocols.

Turing machines are the simple and basic model that simulates physically realizable computation models with little loss in efficiency.

Definition: 1.4: Turing Machines

A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, q_0, q_a, q_r)$

1. Q: set of states
2. Σ: input alphabet
3. Γ: tape alphabet, $\sqcup \in \Gamma$ (blank, $\square$), $\Sigma \subset \Gamma$
4. δ: transition function $\delta : Q \setminus \{q_a, q_r\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$ s.t. $\delta(q, a) = (q', a', x)$, where $x \in \{L, R\}$ means moving left or right. Input set size is fixed and finite. If machine is in state q and head over tape square with a, then the machine replaces a with a' and moves to state q'. The head moves to left or right depending on x.
5. q_0, q_a, q_r: initial, accept, reject states, $q_a \neq q_r$

A Turing machine M works on an input string $x \in \Sigma^*$ as follows:

1. Initially, $x = x_1 x_2 \cdots x_n \in \Sigma^*$ appears on the leftmost n squares of the tape. The rest of the tape is blank. The head starts on the leftmost square of the tape.
2. The initial state is q_0.
3. M moves according to δ, continues until q_a or q_r, then it halts. M may not halt in finite time.

Definition: 1.5:

M accepts $x \in \Sigma^*$ if M with input x eventually halts in q_a.
$L(M) = \{x \in \Sigma^* : M \text{ accepts } x\}$ are the languages accepted by M.

There are different variations/formalizations of Turing Machines:

- Have k tapes (k is a fixed constant) with fixed tape for input
- Tape can be infinite in both directions
- Oblivious Turing machine: movement of head depends on input length only
- Different alphabet

Theorem: 1.1: Properties of Turing Machines

Robustness: Each model can simulate each other with at most polynomial slow down.
Time: a Turing machine runs in $T(n)$ time if it performs at most $T(n)$ basic operations (transitions) on inputs of length n.
Space: a Turing machine runs in $S(n)$ space if it uses at most $S(n)$ spaces on the tape for inputs of length n.
Any Turing Machine can be represented by a string in $\{0, 1\}^*$.

Definition: 1.6: Universal Turing Machine

Every string in $\{0,1\}^*$ can represent a Turing machine. Let M_α be the Turing machine represented by string α.
There exists a universal Turing machine U s.t. U can simulate any other Turing machine given the bit representation.
i.e. Given input (x, α), U can simulate the behavior of M_α on x.

Theorem: 1.2:

If the running time of M_α on x is $T(|x|)$, then the running time of U is $T(|x|) \log T(|x|)$.

Theorem: 1.3:

There exist functions that cannot be computed by any Turing machine.

Proof. The proof is analogous to the diagonalization proof that $\mathbb{R}$ is uncountable.
Consider the function $\mathrm{UC} : \{0,1\}^* \to \{0,1\}$. Let ith string be the description of ith Turing machine M_i.
Given input $x \in \{0,1\}$, if $M_x(x) = 1$, then $\mathrm{UC}(x) = 0$, else $\mathrm{UC}(x) = 1$.
Suppose UC is computable, then $\exists \alpha$ s.t. UC is computed by M_α. *i.e.* $M_\alpha(x) = \mathrm{UC}(x)$ for any x. Then $M_\alpha(x) = \mathrm{UC}(x)$. Contradiction. □

Note: The set of all Turing machines is countable.

Theorem: 1.4: HALT Problem

Let $\mathrm{HALT}(\alpha, x) = 1 \Leftrightarrow M_\alpha(x)$ halts in a finite number of steps. Then HALT is not computable by any Turing machines.

Proof. Suppose $\exists$ a Turing machine M_{HALT} which computes HALT. We can use M_{HALT} to compute TM computing UC in Theorem 1.3.
Define M_{UC}: Given input α, run M_{HALT} on (α, α). If it does not halt, output 1. Otherwise, use a universal Turing machine to compute $b = M_\alpha(\alpha)$ and output $1 - b$
$\mathrm{UC} \leq \mathrm{HALT}$. □

1.2 Time Complexity

Definition: 1.7: Time Complexity of Language

$\mathrm{DTIME}(T(n))$ is the set of all languages $L \subset \{0,1\}^*$ accepted by a Turing maching with running time at most $cT(n)$ on inputs of length n, where c is a contant.

Example: $\mathrm{DTIME}(n^2) = \{$all $L \subset \{0,1\}^*$ accepted by some Turing machine in cn^2 time.$\}$.

Definition: 1.8: Complexity Class P

The polytime class $P = \bigcup_{c \geq 1} \mathrm{DTIME}(n^c)$ is the set of decision problems that can be easily solved (in polynomial time) by a Turing machine.

Theorem: 1.5: Linear Speedup Theorem

Given any $c > 0$ and any k-tape Turing machine solving a problem in time $T(n)$, there exists another k-tape Turing machine that solves the same problem in time at most $\frac{T(n)}{c} + 2n + 3$.

Definition: 1.9: Complexity Class NP

$L \subset \{0,1\}^*$ is NP (Nondeterministic Polynomial time) if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time Turing machine M, the verifier for L s.t. $\forall x \in \{0,1\}^*$, $x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}$ s.t. $M(x,y) = 1$.
y is called a certificate/witness for x w.r.t. L and M.

Definition: 1.10: Complexity Class EXP

$\text{EXP} = \bigcup_{c \geq 1} \text{DTIME}(2^{n^c})$ is the set of decision problems that can be solved in exponential time by a Turing machine.

P $\subset$ NP $\subset$ EXP. NP $\subset$ EXP, because we can brute force all NP problems by trying all possible inputs which is exponential time w.r.t. input size.

Theorem: 1.6: Time Hierarchy Theorem

Let f, g be time constructable functions s.t. $f(n)\log(f(n)) = o(g(n))$. Then $\text{DTIME}(f(n)) \subsetneq \text{DTIME}(g(n))$.
e.g. $\text{DTIME}(n) \subsetneq \text{DTIME}(n^{10})$

Proof. Consider the function (language) L s.t. given an input α of length $f(n)$. If $M_\alpha(\alpha)$ (by some universal Turing machine) accepts within $f(n)\log f(n)$ steps, then reject, otherwise accept. Since $f(n)\log(f(n)) = o(g(n))$, $L \in \text{DTIME}(g(n))$.
Suppose there exists a Turing machine M_i that decides L in time $cf(n)$, where c is a constant, $|M_i| = f(n)$. We can assume that $|M_i|\log|M_i| > c|M_i|$ by padding with zeros, even though M_i is a fixed length string. Then $M_i(\langle M_i \rangle)$ rejects if and only if accepts. Because we can always decide to accept or reject within $c|M_i| < |M_i|\log|M_i|$ steps. Once we decide to accept, the TM should actually reject it. ☐

1.3 Reduction and NPC

Definition: 1.11: Karp Reductions

$L \leq_p L'$ if there exists a poly time TM M s.t. $x \in L \Leftrightarrow M(x) \in L'$.

Definition: 1.12: NP-Hard and NPC

$L' \in P \Rightarrow L \in P$
$L' \in$NP-Hard if $L \leq_p L'$ for all $L \in$NP
$L \in$NPC if $L \in$NP and $L \in$NP-Hard.

Definition: 1.13: Boolean Formula

A boolean formula φ consists of n variables $u_1, u_2, ..., u_n$ with logical operators $\vee, \wedge, \neg$. Let $z \in \{0,1\}^n$, $\varphi(z)$ denotes the truth value of φ when $u_i = z_i$.

Example: $\varphi = (u_1 \wedge u_4) \vee (\neg u_3 \vee u_3) \vee (u_4 \wedge u_7)$ is a boolean formula

Definition: 1.14: CNF and SAT

A boolean formula is in CNF form if it is an AND of ORs of variables and their negations (literals).
A k-CNF is a CNF formula in which every clause contains at most k variables.
k-SAT is the language of satisfiable k-CNF formulae: $\{k\text{-CNF}\varphi : \exists \text{satisfying assignment for } \varphi\}$
SAT $= \{\varphi : \varphi \text{ is a CNF formula which has a satisfying assignment}\}$

Example: $(u_1 \vee u_2 \vee \neg u_5) \wedge (u_1 \vee u_7) \wedge (\neg u_3 \vee \neg u_5)$ is a CNF.
3-CNF: every clause has at most three literals.
3-SAT: language of satisfiable 3-CNF formulae.

Definition: 1.15: TM-SAT

The Turing machine SAT problem is defined as

$$\text{TM-SAT} = \left\{\left\langle \alpha, x, 1^n, 1^t \right\rangle : \exists y \in \{0,1\}^n \text{ s.t. } M_\alpha \text{ outputs 1 on } \langle x, y \rangle \text{ within } t \text{ steps}\right\}$$

Here, 1^n represents the string of n 1s, 1^t represents the string of t 1s.

Definition: 1.16: Boolean Circuit

A boolean circuit with n inputs and 1 output is a directed acyclic graph with n sources and 1 sink, all non-source vertices are labelled $\wedge, \vee, \neg$, where $\wedge, \vee$ have fan-in of 2 and $\neg$ has fan-in of 1.
The size of a circuit is $|C|$, which is the number of vertices in C.
Given input $x \in \{0,1\}^n$, the circuit outputs $C(x)$.

Theorem: 1.7: Universality of $\wedge, \vee, \neg$

For every Boolean function $f : \{0,1\}^L \to \{0,1\}$ (total of 2^{2^L} functions), there is an L-CNF formula φ of size at most $\mathcal{O}(l2^l)$ s.t. $\varphi(u) = f(u)$ for all $u \in \{0,1\}^L$. *i.e.* $\forall f$, there is a boolean circuit computing f.

Proof. For every $v \in \{0,1\}^L$, there is a clause c_v in L variables s.t. $c_v(v) = 0$ and for all other inputs u, $c_v(u) = 1$. *e.g.* $c_v(z_1, ..., z_L) = \bar{z_1} \vee z_2 \vee \cdots \vee z_L$ if $v = (1, 0, ..., 0)$
Let φ be AND of all clauses c_v s.t. $f(v) = 0$. ☐

Theorem: 1.8: SAT$\leq_p$3-SAT

There exists a transformation mapping any CNF φ to 3-CNF ψ s.t. ψ is satisfiable $\Leftrightarrow$ φ is satisfiable.

Proof. Suppose φ is a 4-CNF, $c = (u_1 \vee \bar{u_2} \vee \bar{u_3} \vee u_4)$.
Add a new variable z and replace c with $c_1 \wedge c_2$, where $c_1 = u_1 \vee \bar{u_2} \vee z$, $c_2 = \bar{u_3} \vee u_4 \vee \bar{z}$.
Similarly for k-CNF clauses, replace with $(k-1)$-CNF clause and 3-CNF clause and then recurse. ☐

Lemma: 1.1:

For every $T(n)$ time Turing machine M, there exists an $\mathcal{O}(T(n)^2)$ size circuit family $\{C_n\}_{n\in\mathbb{N}}$ s.t. $C_n(x) = M(x)$, $\forall x \in \{0,1\}^n$.

Proof. Firstly, simulate M by an $\mathcal{O}(T(n)^2)$ time oblivious Turing machine M'.
Given an input x to M'. Let $z_1, z_2, ..., z_k$ be the local snapshots of the computation of M' on x.
z_i is the encoding of state at time i and symbol read by head. z_i is a constant size binary string.
z_i only depends on z_{i-1} and $z_{i'}$ where i' is the last time when head is in the same position, or some symbol of x. z_i can be computed from previous snapshots on x using a constant size circuit.
The composition of these circuits gives the final circuit from x to z_k, where $k = \mathcal{O}(T(n)^2)$ □

Theorem: 1.9: Cook-Levin Theorem

3-SAT is NPC.

Proof. In this proof, we show that for any language $L \in$NP, $L \leq_p$ TM-SAT $\leq_p$ circuit-SAT $\leq_p$ 3-SAT.

1. $L \leq_p$ TM-SAT
Since $L \in$NP, there exists polynomial p and Turing machine M s.t. $x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}$ s.t. $M(x,y) = 1$ and M runs in time $q(n)$ for some polynomial q.
Consider the map $x \to \langle \lfloor M \rfloor, x, 1^{p(|x|)}, 1^{q(m)} \rangle$, where $m = |x| + p(|x|)$ (reading the input + verifying). This map is a poly time reduction and $x \in L \Leftrightarrow \langle \lfloor M \rfloor, x, 1^{p(|x|)}, 1^{q(m)} \rangle \in$TM-SAT.

2. TM-SAT $\leq_p$ circuit-SAT
Consider $\langle \lfloor M \rfloor, x, 1^n, 1^t \rangle$. Consider a Turing machine $\hat{M}$ which on input y of length n runs M on (x, y) for t steps and accepts iff M accepts.
By Lemma 1.1, we can construct the $\mathcal{O}((t+n)^2)$ size circuit C_n s.t. $C_n(y) = \hat{M}(y), \forall y \in \{0,1\}^n$.
Then C_n is satisfiable $\Leftrightarrow \langle \lfloor M \rfloor, x, 1^n, 1^t \rangle \in$TM-SAT.

3. Circuit-SAT $\leq_p$ 3-SAT
Let $v_1, v_2, ..., v_m$ denote the nodes of the circuit C. For each node v_i, introduce variable u_i. If $v_i = v_j \wedge v_l k$, add clauses corresponding to $u_i = u_j \wedge u_k$, *i.e.* $(\bar{u_i} \vee \bar{u_j} \vee u_k) \wedge (\bar{u_i} \vee u_j \vee \bar{u_k}) \wedge (\bar{u_i} \vee u_j \vee u_k) \wedge (u_i \vee \bar{u_j} \vee \bar{u_k})$.
We can define similar clauses for $v_i = v_j \vee v_k$. If $v_i = \neg v_j$, we add claused for $u_i = \bar{u_j}$ by $(u_i \vee u_j) \wedge (\bar{u_i} \vee \bar{u_j})$.
If v_i is the final output node, then add u_i to the formula.
We then obtain a 3-CNF formula φ which will be polynomial time reduction (constant addition for each node)
Circuit C is satisfiable $\Leftrightarrow \varphi$ is satisfiable. □

Importance of 3-SAT: it is useful in reduction, mathematical logic, constraint satisfaction problem, and it is well-studied.

Theorem: 1.10:

0/1-Integer Programming: Given m linear inequalities with rational coefficients and n variables $u_1, u_2, ..., u_n$. Is there an assignment of 0s and 1s satisfying all inequalities?
0/1-Integer Programming is NP Complete.

Proof. 0/1-Integer Programming is NP:
Certificate: an assignment to $u_1, ..., u_n$. Plug in and check in polynomial time

3-SAT$\leq_p$ 0/1-Integer Programming:
Reduction: express CNF formula as an integer program by expressing each clause as an inequality:
e.g. $u_1 \vee \bar{u_2} \vee \bar{u_3} \to u_1 + (1 - u_2) + (1 - u_3) \geq 1$ □

1.4 co-NP and co-NPC

Definition: 1.17: co-NP

For $L \subset \{0,1\}^*$, define $\bar{L} = \{0,1\}^* \setminus L$, co-NP$= \left\{L : \bar{L} \in NP\right\}$.
Alternatively, $L \subset \{0,1\}^*$ is in co-NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and polynomial time Turing Machine M s.t. $\forall x \in \{0,1\}^*$, $x \in L \Leftrightarrow \forall y \in \{0,1\}^{p(|x|)}$, $M(x, y) = 1$.

Definition: 1.18: co-NPC

A language is co-NP complete if it is in co-NP and every co-NP problem can be reduced to it.

1.5 Non-deterministic Turing Machine

Definition: 1.19: Non-deterministic Turing Machine

A non-deterministic Turing Machine (NDTM) (not realized) has two transition functions δ_0 and δ_1. When NDTM computes a function, at each step, it makes an arbitrary choice as to which transition function to apply. $M(x) = 1$ if there exists some sequence of choices (the non-deterministic choices which would make M reach q_a on input x). M runs in time $T(n)$ if $\forall x \in \{0,1\}^*$ and every non-deterministic choices, M halts or reaches q_a within $T(n)$ steps.

Definition: 1.20: NTIME and NP

Given $T : \mathbb{N} \to \mathbb{N}$, $L \subset \{0,1\}^*$, $L \in$ NTIME$(T(n))$ if there exists $c > 0$ and $cT(n)$ time NDTM M s.t. $\forall x \in \{0,1\}^*$, $x \in L \Leftrightarrow M(x) = 1$.
NP $= \bigcup\limits_{k \geq 1}$ NTIME(n^k).

We can simulate a NDTM using a DTM if the sequence of choices (certificate) is known.

Theorem: 1.11: Non-deterministic Time Hierarchy Theorem

Let f, g be time constructible functions s.t. $f(n+1) = o(g(n))$, then NTIME$(f(n)) \subsetneq$ NTIME$(g(n))$.

Many natural problems in NP are NP-complete, but not every problem in NP is in either P or NPC. *e.g.* Factoring is in NP, but Factoring is not NPC and not P.

Theorem: 1.12: Ladner's Theorem

If P$\neq$NP, then there exists a language $L \in NP \setminus P$ that is not NP-complete.

Proof. For $H : \mathbb{N} \to \mathbb{N}$, let 3-SAT$_H$ $= \left\{\varphi \circ 1^{H(n)} : \varphi \in \text{3-SAT and } n = |\varphi|\right\}$ be the length n satisfiable formulas padded with $H(n)$ 1s.
If H grows fast, *e.g.* $H(n) = 2^n$, then 3-SAT$_H \in P$. (The length of formula is logarithm w.r.t. length of input). Then brute force is poly time w.r.t. length of inputs.
We need to find H that doesn't grow too fast to make it in P and not too slow to make it in NPC

Lemma 1. *If H is polytime computatble, then 3-SAT$_H$ $\in$NP.*

Lemma 2. *If $H = n^{\omega(1)}$, e.g. $H = n^{\log n}$, then 3-SAT$_H$ is not NP-complete unless P=NP.*

Proof. If it is NP-Hard, then 3-SAT$\leq_p$ 3-SAT$_H$.
We start from a 3-SAT formula φ and get a new formula φ' which is smaller by a polynomial factor s.t. φ is satisfiable $\Leftrightarrow$ φ' is satisfiable.
Repeat this process, we can reduce size φ' to constant and brute force the solution to solve 3-SAT in polynomial time.
Contradiction. Thus 3-SAT$_H$ cannot be NP-Hard. □

Lemma 3. *If P$\neq$NP, then there exists H that is polytime computatble grows superpolynomially, but 3-SAT$_H$ $\notin$ P*

Lemma 4. *A modification of previous lemma. Suppose P$\neq$NP in a meaningful way. i.e. there exists a polynomial time computable function t s.t. $t = n^{\omega(1)}$ and 3-SAT requires time $t^{\omega(1)}$, then 3-SAT$_t$ $\notin$P.*

Proof. If 3-SAT$_t$ $\in$P, then 3-SAT is in time $t^{O(1)}$. □

By the lemmas, we get that there exists intermediate problems in NP. □

1.6 Relativization

Can we use diagonalization to prove P$\neq$NP?

***Definition:* 1.21: Diagonalization**

Proofs that only uses ability of Turing machiens to simulate other Turing machines.

***Definition:* 1.22: Oracle Turing Machines**

Turing machines are given access to an oracle that can solve the decision problem for some language $O \subset \{0,1\}^*$.
They have a special oracle tape: on it, write $q \in \{0,1\}^*$ and in one step, it gets the answer to "is q in O?" This can be repeated arbitrarily often.

***Definition:* 1.23: P^O and NP^O**

P^O is the set of languages decided by a poly time Turing machine with oracle access to O.
NP^O is the set of languages decided by a poly time non-deterministic Turing machine with oracle access to O.

Example: SAT$\in$ P$^{\text{SAT}}$, NP$\subset$ P$^{\text{SAT}}$, P$\subset$ P^L for L $\in$NPC.
$\overline{\text{SAT}} \in$ P$^{\text{SAT}}$, co-NP$\subset$ P$^{\text{SAT}}$

Theorem: 1.13: Properties of Oracle Turing Machines

1. Oracle TMs can also be represented as strings
2. There exists a universal Turing Machine with oracle access to O
3. Proofs using representation of TMs as strings and simulation of TMs by other TMs also hold for oracle TMs. *i.e.* they relativize.

Proofs by diagonalization relativize.

Theorem: 1.14: Baker-Gill-Solovay

There exist oracles A, B s.t. $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$

Proof. Let $\mathrm{EXP} = \bigcup_c \mathrm{DTIME}(2^{n^c})$. $\mathrm{EXPCOM} = \{\langle M, x, 1^n \rangle : M$ outputs 1 on x in 2^n steps$\}$.

$\mathrm{EXP} \subset \mathrm{P}^{\mathrm{EXPCOM}} \subset \mathrm{NP}^{\mathrm{EXPCOM}} \subset EXP$. The final inclusion comes from that NP can be done in exponential time and brute force the solution is exp time.
Thus for A=EXPCOM, $\mathrm{P}^A = \mathrm{NP}^A$.

For language B, let $U_B = \{1^n : \exists x \in B \text{ s.t. } |x| = n\}$ be the unity language of B.
It is easy to verify that $U_B \in \mathrm{NP}^B$ for all B. To check $x = 1^n \in U_B$, make non-deterministic guess of string of length n and query the oracle.
We then find B s.t. $U_B \notin \mathrm{P}^B$ by diagonalization.
Enumerate $M_1, M_2, ...$ poly time TMs with oracle tapes. Each TM appears infinitely often.
Construct B in stages. Initially B is empty and we add strings at each string.
In stage i, make sure M_i^B does not decide U_B in $\frac{2^n}{10}$, so not in poly time.
Choose n larger than all strings whose status has been decided, so previous machines cannot decide it.
Note $U_B(1^n)$ does not depend on anything decided in the past.
We want to choose B s.t. $M_i(1^n)$ messes up.
Run M_i on input 1^n for $\frac{2^n}{10}$ steps. All queries decided in the past answer correctly. For all new queries, answer "Not in B".
After M_i finishes and outputs 0/1. Make sure M_i messes up.
Now M_i could have queried at most $\frac{2^n}{10}$ strings in $\{0,1\}^n$.
If M_i accepts, then all strings in $\{0,1\}^n$ are not in B.
If M_i rejects, pick a string in $\{0,1\}^n$ not queried and add it to B.
Then $M_i^B(1^n)$ messes up. M_i^B cannot compute U_B. □

2 Space Complexity

Let M be a deterministic Turing machine (not necessarily halting)

Definition: 2.1: Space Complexity

Let $S : \mathbb{N} \to \mathbb{N}$ and $L \subset \{0,1\}^*$. $L \in \text{SPACE}(s(n))$ if there exists a constant c and Turing machine M deciding L s.t. on every input $x \in \{0,1\}^*$, the total number of locations that are at some ppint non-blank during M's execution on x is at most $c \cdot s(|x|)$. (Input doesn't count)

Theorem: 2.1:

$\text{NTIME}(t(n)) \subset \text{SPACE}(t(n))$

Proof. Try all possible computation paths of $t(n)$ steps for an NDTM on input of length n. This can be done in $\mathcal{O}(t(n))$ space, by additionally storing $t(n)$ transitions to keep track of current branch. □

Definition: 2.2: Space Complexity Classes

$\text{SPACE}(f(n)) = \{L : L \text{ decided by a TM with } \mathcal{O}(f(n)) \text{ space complexity}\}$
$\text{NSPACE}(f(n)) = \{L : L \text{ decided by a NDTM with } \mathcal{O}(f(n)) \text{ space complexity}\}$
$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k)$. It formalizes problems solvable by computers with bounded memory.
$\text{NPSPACE} = \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k)$

By Theorem 2.1, P $\subset$ NP $\subset$ PSPACE $\subset$ EXPTIME.

Let M be halting TM with space complexity $f(n)$. Then the time complexity has lower bound $f(n)$. With exponential time $2^{\mathcal{O}(f(n))}$, we can simulate PSPACE.
The number of steps is at most the number of possible configurations. If configuration repeats, then machine loops.

Theorem: 2.2:

P $\neq$ EXPTIME

The proof is from time hierachy theorem (Theorem 1.6).
Thus either P$\neq$NP or NP$\neq$PSPACE or PSPACE$\neq$EXPTIME.

Theorem: 2.3: Savitch's Theorem

For any function $f(n)$, where $f(n) > n$, $\text{NSPACE}(f(n)) \subset \text{SPACE}(f(n)^2)$.

Proof. Let N be a NDTM with space complexity $f(n)$. Consider a deterministic TM that tries every branch of N. Each branch uses at most $f(n)$ space. Thus total space $\leq 2^{2^{\mathcal{O}(f(n))}}$ (which is based on number of branches)

Let $N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ be a NDTM with space complexity $f(n)$.
We will construct a deterministic TM M with space complexity $\mathcal{O}(f(n)^2)$ s.t. $L(M) = L(N)$.
Let $w \in \Sigma^n$ be the input of length n. Define $G = (V, E)$ the $f(n)$ space configuration graph of N, where $V = \{\text{configurations of } N \text{ with at most } f(n) \text{tape symbols}\}$

$E = \{(c_1, c_2) \in V \times V : c_1 \text{ yields } c_2\}$
Since N is NTM, c_1 can have edges to multiple configs.
$|V| \leq |Q|f(n)|\Gamma|^{f(n)}$. Fix d s.t. $|V| \leq 2^{df(n)}$ for any n.

$w \in L(N) \Leftrightarrow$ there exists a path in G of length at most $2^{df(n)}$ from $q_0 w$ to an accepting configuration of the form $xq_a y$
We want to find a deterministic algorithms .t. it finds a path from q_0 to q_1.
Define $\textsc{CanYield}(c_1, c_2, t)$ for $c_1, c_2 \in V$, $t \geq 1$ a recursive algorithm s.t.
If $t = 1$, accept if c_1 yields c_2, otherwise reject
If $t \geq 2$, for each $c_3 \in V$, recurse on $\textsc{CanYield}(c_1, c_3, \lceil \frac{t}{2} \rceil)$ and $\textsc{CanYield}(c_3, c_2, \lfloor \frac{t}{2} \rfloor)$. If both accept for some c_3, then accept, otherwise reject.

We set $t = 2^{df(n)}$, if $\textsc{CanYield}$ accept, then the Turing maching M accepts.
$\textsc{CanYielf}(c_1, c_2, t)$ has t levels of recursion. Each level of recursion uses $\mathcal{O}(f(n))$ additional space.
Total space is $\mathcal{O}(t(f(n)))$, but $t = \mathcal{O}(f(n))$, so total space is $\mathcal{O}((f(n))^2)$. □

Following the above PSPACE=NPSPACE.

Definition: 2.3: PSPACE-Complete

A language B is PSPACE-Complete if
1. B is PSPACE
2. Every A in PSPACE is polytime reducible to B (B is PSPACE-hard)

Theorem: 2.4:

If B is PSPACE-Complete and $B \in$P, then P=PSPACE. If $B \in$NP, then NP=NPSPACE.

Definition: 2.4: Fully Quantified Boolean Formula

A fully quantified boolean formula is a boolean formula where every variable in the formula is quantified $(\exists, \forall)$ at the beginning of the formula (prenex normal form).

Example: $\forall x, \exists y\, [(x \vee y) \wedge (\bar{x} \vee \bar{y})]$ is true.
$\exists x, \exists y\, [x \vee \bar{y}]$ is true.
$\forall x[x]$ is false.

Definition: 2.5: TQBF

The language TQBF$= \{\varphi : \varphi \text{ is a true fully quantified boolean formula}\}$

SAT is a special case where all quantifiers are $\exists$. TAUT is a special case where all quantifiers are $\forall$.
SAT$\leq_p$TQBF, TAUT$\leq_p$TQBF.

Theorem: 2.5: Meyer-Stockmeyer

TQBF is PSPACE-Complete

Proof. 1. TQBF$\in$PSPACE
Let φ be a fully quantified boolean formula as input to a Turing machine T
If φ has no quantifiers, evaluate φ, accept if it evaluates to 1.
If $\varphi = \exists x \psi$, the recursively call T on ψ with $x = 0$ and then $x = 1$, if either result in accept, then accept

φ, else reject.
If $\varphi = \forall x\psi$, then recursively call T on psi. Accept if both $x = 0$ and $x = 1$ accept.
Note that space can be reused, for computing $\psi_{x=0}$ and $\psi_{x=1}$. Thus the space the algorithm used is $s_{n,m} = s_{n-1,m} + \mathcal{O}(m)$, $s_{n,m} = \mathcal{O}(nm)$ where n is the number of variables, and m is the description size.

2. TQBF is PSPACE-Hard
For all $A \in$PSPACE, there exists a constant k and Turing machine M that decides A in space $\leq n^k$.
We find a polytime reduction from any string w to a fully quantified φ that simulates M on w.
Given $\mathcal{O}(n^k)$ possible configurations, each needs time $2^{\mathcal{O}(n^k)}$ to simulate on a DTM.
Fix M and w, the goal is to construct a QBF φ which is true if and only if M accepts w.
Let $\varphi_{c_1,c_2,t}$ be a formula which is true if and only if M can go from c_1 to c_2 in at most t steps.
Then set $c_1 = c_{\text{start}}$, $c_2 = c_{\text{accept}}$, $t = 2^{dn^k}$.
If $t = 1$, encode $c_1 = c_2$ on c_1 yields c_2.
If $t > 1$, $\varphi_{c_1,c_2,t} = \exists m_1 \left[\varphi_{c_1,m_1,\frac{t}{2}} \wedge \varphi_{m_1,c_2,\frac{t}{2}}\right]$, m_1 represents configuration of M.
But at each t, φ gets doubled. So we consider the following reconstruction:
$\varphi_{c_1,c_2,t} = \exists m_1 \forall c_3, c_4 \left[((c_3, c_4) = (c_1, m_1)) \vee ((c_3, c_4) = (m_1, c_2)) \Rightarrow \varphi_{c_3,c_4,\frac{t}{2}}\right]$
The final formula size is $\mathcal{O}(n^{2k})$ for $t = 2^{dn^k}$ (There are $\mathcal{O}(n^k)$ recursions, and for each level of the recursion, formula size increased by $\mathcal{O}(n^k)$) □

PSPACE captures the complexity of several 2-player games of perfect information.

2.1 Sublinear Spaces

Class L and NL are sublinear space bounds where $f(n)$ is much smaller than n.
We consider a 2-tape TM

1. read-only input tape
2. read/write work tape (on which we measure the space complexity)

We don't have to store all data on the main memory

Definition: 2.6: L and NL

L is the class of languages decidable in log space on a DTM, L = SPACE($\log n$)
NL = NSPACE($\log n$)

Example: $A = \left\{0^k 1^k : k \geq 0\right\}$ is a language in L.

For log space problems, #configs$\leq 2^{\mathcal{O}(\log n)} = n^{\mathcal{O}(1)}$, so the TM can run in poly time.
$L \subset P$, similarly $NL \subset P$.

Definition: 2.7: PATH

$$\text{PATH} = \{\langle G, s, t\rangle : G \text{ directed graph with a directed path } s \to t\}$$
$$\overline{\text{PATH}} = \{\langle G, s, t\rangle : G \text{ directed graph without a directed path } s \to t\}$$

PATH$\in$P, also PATH$\in$NL (Gues the next vertex iteratively and save the current vertex only. If it reaches t, then accept) $\overline{\text{PATH}}$ $\in$coNL

Conjecture: NL$\neq$L.

Definition: 2.8: NL-Complete

A is NL-Complete if $A \in$NL and $\forall B \in$NL, $B \leq_{\text{log-space}} A$.
We require that every bit of output is computable in log space.
Log space transducer: a TM with 3 tapes

1. Read only input tape
2. Write only output tape (head cannot move left)
3. Read/Write work tape with $\mathcal{O}(\log n)$ symbols.

The transducer computes a function $f : \Sigma^* \to \Sigma^*$ where $f(w)$ written on output tape when M halts with w on input tape. f is a log space computable function.
A is logspace reducible to B ($A \leq_L B$) if A is mapping reducible to B by a logspace computable function.

Theorem: 2.6:

If $A \leq_L B$ and $B \in L$, then $A \in L$

Proof. Can define a logspace algo for A on input w and compute $f(w)$ the logspace reduction. Then apply logspace algo for B, but storage of $f(w)$ is too large to fit in logspace.
So we compute every symbol of $f(w)$ needed when we need it for B. Each symbol can be computed in logspace □

Corollary 1. *If NL-Complete are in L, then NL=L.*

Theorem: 2.7:

PATH is NL-Complete

Proof. PATH is NL: non-deterministically guess vertices on $s - t$ path.

PATH is NL-hard: $\forall A \in$NL, $A \leq_L$PATH.
For M_A and a string w, the configuration graph has poly n vertices and poly n edges $(c_1, c_2) \in E$ if c_1 yields c_2.
For every pair of vertices c_1, c_2, brute force to check if there is an edge in log space.
M accepts $w \Leftrightarrow$ there is path from c_s to c_a. □

$NL \subset P$.

Proof. Any TM that uses space $f(n)$ runs in time $n2^{\mathcal{O}(f(n))}$. Log space transducer runs in polytime.

If $A \in$NL, then $A \leq_L$PATH. So it sufficies to show that PATH$\in$P, which is true. □

Definition: 2.9: NL Certificate

$A \in$NL if there exists a DTM with additional read once (head can only move right) input tape and $p : \mathbb{N} \to \mathbb{N}$ s.t. $\forall x \in \{0,1\}^*$, $x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}$ s.t. $M(x, y) = 1$ where x is an input tape, y is a special read-once tape and M uses at most $\mathcal{O}(\log |x|)$ space on read/write tape for all x.

Equivalence: NDTM can simulate DTM by Nondeterministically guess what the head is reading. If in NDTM, there is an accepting sequence, take it as the certificate on the R/W tape. DTM can simulate.

Theorem: 2.8: Immerman-Szelepcsenyi

NL=coNL

Proof. We show that coNL⊂NL by showing that $\overline{\text{PATH}}$ ∈NL and NL⊂coNL by showing that PATH∈co-NL.

$\overline{\text{PATH}}$: accept ⇔ input graph does not have a path from s to t.

We need to find a read once proof and a logspace TM that verifies it.
Easier problem: Let C be the number of nodes reachable from s. Assume M knows C.
Given G, s, t, c, M goes over all m nodes of G. For each node u, M guesses if u is reachable from s. If yes, then it can be verified by NDTM guessing the path, update the counter.
Suppose counter reaches C over all nodes and t does not contribute to the count, then accept, else reject.
Increasing the counter ensures that the same path cannot appear twice

General problem: If M doesn't know C.
Let $A_i = \{\text{nodes at distance } i \text{ or less from } s\}$. $C_i = |A_i|$, $C = |A_m|$ all nodes reachable from S in m steps.
We want to compute C_{i+1} from C_i.
Claim: Given C_i, $\forall v$, we can verify if $v \in A_{i+1}$ and also $v \notin A_{i+1}$.
We simply give the path and check the length, we can verify $v \in A_{i+1}$. If $C_i = |A_i|$, then $\exists s_1, s_2, ..., s_{C_i}$ with path length $\leq i$. Write down the path, verifier checks that these are valid distinct paths. $v \notin A_{i+1}$ if $v \neq s_i$ and v is not adjacent to any s_i.

For $v_1, v_2, ..., v_m$, vertices in increasing order, concatenate all certificates. M maintains a counter and compute $|A_{i+1}| = C_{i+1}$ □

3 Polynomial Hierarchy

Definition: 3.1: Independent Sets

independent sets: set of vertices s.t. all vertices are not connected by an edge

$$\text{IDTSET} = \{\langle G, k\rangle : G \text{ has an independent set of size } k\}$$
$$\text{EXACT-IDTSET} = \{\langle G, k\rangle : G \text{ has the largest independent set of size } k\}$$

IDTSET∈NP, EXACT-IDTSET∈PSPACE, since we can check subsets of size k and $k+1$ using poly space.

Definition: 3.2: Σ_2^p

Σ_2^p is the set of all languages L s.t. there exists poly time Turing machine M and polynomial q s.t. $x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)} \forall v \in \{0,1\}^{q(|x|)}, M(x,u,v) = 1$

When we remove v, we get NP. When we remove u, we get co-NP. So Σ_2^p is a stronger notation.

EXACT-INDSET∈ Σ_2^p. There exists a vertex set of size k s.t. this set is an independent set and for all other independent sets, the size is smaller than k.

Definition: 3.3: Σ_i^p

For all i, we define Σ_i^p, $L \in \Sigma_i^p$ if there exists a poly time TM M and polynomial q s.t. $x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \cdots Q_i u_1 \in \{0,1\}^{q(|x|)}$, $M(x, u_1, ..., u_i) = 1$, Q_i is for all if i even, exists if i odd.

Note: $\forall i, \Sigma_i^p \subset$PSPACE, $\Sigma_0^p =$P, $\Sigma_1^p =$NP.

Definition: 3.4: Π_i^p

$\Pi_i^p =$co-$\Sigma_i^p = \left\{\bar{L} : L \in \Sigma_i^p\right\}$
$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \cdots Q_i u_1 \in \{0,1\}^{q(|x|)}$, $M(x, u_1, ..., u_i) = 1$, Q_i is exists if i even, for all if i odd.

Definition: 3.5: Polynomial Hierarchy

PH$=\bigcup_i \Sigma_i^p$.

Theorem: 3.1: Properties of Polynomial Hierarchy

1. $\Sigma_i^p \subset \Pi_{i+1}^p \subset \Sigma_{i+2}^p$
2. PH$=\bigcup_i \Pi_i^p$
3. $\Sigma_i^p \subset \Sigma_{i+1}^p \subset \Sigma_{i+2}^p$.

All the subset containments are strict. The polynomial hierarchy does not collapse.
If $\Sigma_i^p = \Pi_i^p$, then the PH collapses to i-th level.

Definition: 3.6: Time Space

TISP$(f(n), g(n))$ are the languages L s.t. a single TM M decides L in time $f(n)$ and space $g(n)$.

Theorem: 3.2: Time Space Trade-off

NTIME$(n) \not\subset$TISP$(n^{1.2}, n^{0.2})$.

Note: SAT can be solved by NDTM in linear time, but cannot solve it within limitation in both time and space.

4 Boolean Circuit

Definition: 4.1: Boolean Circuit

A circuit is a directed acyclic graph with n inputs and 1 output. All non-input vertices are gates $\vee, \wedge$ with fan-in 2, and $\neg$ with fan-in 1.
The size of a circuit C is the number of vertices in it. The number of edges is constant multiple of number of vertices.

Recall that the Circuit-SAT is NP-Complete

Definition: 4.2: $T(n)$-size Circuit Family

$T(n)$-size circuit family is a sequence of boolean circuits $\{C_n\}_{n\in\mathbb{N}}$ s.t. $|C_n| \leq n, \forall n$.
L is in SIZE($T(n)$) if there exists a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ s.t. $\forall x \in \{0,1\}^n, x \in L \Leftrightarrow C_n(x) = 1$.

Lemma: 4.1:

For every $T(n)$ time TM M, there exists $\mathcal{O}(T(n)^2)$ size circuit family $\{C_n\}_{n\in\mathbb{N}}$ s.t. $C_n(x) = M(x), \forall x \in \{0,1\}^n$.
$L \in \text{TIME}(T(n)) \Rightarrow L \in \text{SIZE}(\mathcal{O}(T(n)^2))$

Definition: 4.3: P/poly

P/poly is the class of languages decidable by poly size circuits.
P/poly= $\bigcup_c \text{SIZE}(n^c)$

P⊂P/poly, all languages decidable by TM in poly time can be decided by poly size circuits

1. any unary language $1^n \in$P/poly
2. there exists undecidable unary languages (number of TMs are countable, but number of unary languages is uncountable). *e.g.* $\{1^n : n\text{'s binary expansion encodes } \langle M, x\rangle \text{ s.t. } M \text{ halts on } x\}$

Definition: 4.4: Non-uniform

Let $L \subset \{0,1\}^*$ be a language, $L_n \subset L \cap \{0,1\}^n$ can have different algorithm A_n for L_n.

To prove P≠NP, it sufficies to find a function in NP\P/poly. Is it possible that P≠NP, but NP⊂P/poly? *i.e.* could SAT have small circuit family deciding it?

Theorem: 4.1:

Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by circuit of size $\mathcal{O}(2^n)$ (best possible is $\mathcal{O}\left(\frac{2^n}{n}\right)$)
Most functions have circuit size $\geq \frac{\epsilon 2^n}{n}$.

Proof. $f(x_1, x_2, ..., x_n) = [(x_1 = 0) \wedge f_0(x_2, ..., x_n)] \vee [(x_1 = 1) \wedge f_1(x_2, ..., x_n)]$.

The number of functions with circuit size S is $\leq S^{\mathcal{O}(S)}$ (specify types of gates, then specify the inputs and outputs)
To describe a circuit of size S, it takes $\mathcal{O}(S \log S)$ bits.

If $S=\frac{\epsilon 2^n}{n}$, the number of functions is $2^{2^n} >> \left(\frac{\epsilon 2^n}{n}\right)^{\frac{\epsilon 2^n}{n}}$, if ϵ is small enough. Thus most funcitons are not computatble and require large circuits. □

Theorem: 4.2: Size Hierarchy Theorem

SIZE(S)$\subsetneqq$ SIZE($S\log S$)

Theorem: 4.3: Karp-Lipton Theorem

If NP$\subset$P/poly, then PH$=\Sigma_2^p$.

Proof. It suffices to show $\Pi_2^p\subset\Sigma_2^p$.
Consider a Π_2^p-complete language: Π_2^p-SAT which contains all true formula of the form $\forall u\in\{0,1\}^n, \exists v\in\{0,1\}^n$ s.t. $\phi(u,v)=1$. (1)
Since NP$\subset$P/poly, there exists a poly size circuit family $\{C_n\}_{n\in\mathbb{N}}$ s.t. for all boolean formula ϕ of size n, $C_n(\phi)=1\Leftrightarrow\phi$ is satisfiable. *i.e.* $\exists v$ s.t. $\phi(v)=1$.
Then, we can find the solution to ϕ using poly size circuit $\exists\{C'_n\}_n\ C'_n(\phi)=v$ s.t. $\phi(v)=1$ if ϕ is satisfiable. If C_n is of size $q(n)$, then C'_n has size $\le 100q(n)^2$.
Suppose $\forall u,\exists v$ s.t. $\phi(u,v)=1$
Let $\phi(u,x)=\phi_u(x)$ (fix u, check if ϕ_u is satisfiable by x) If $\exists v$ s.t. $\phi_u(v)=1$, then C'_n can find it. $\phi_u(C'_n(\phi_u))=1$. *i.e.* exists circuit C'_n s.t. $\forall u$, $\phi_u(C'_n(\phi_u))=1$.
$\exists w\in\{0,1\}^{100q(n)^2}$ s.t. $\forall u\in\{0,1\}^n$, w is a circuit C'_n and $\phi(u,C'_n(\phi_u))=1$ (2)
(1)$\Leftrightarrow$(2), so $\Pi_2^p\subset\Sigma_2^p$ □

5 Randomization

Given a population, a part of them have property P, the other do not. How can we determine the fraction? Deterministically, we can query every individual in the population. But we can also probabilistically estimate and we only need to query a fraction of the population.

Given two polynomial circuits that evaluates p_1, p_2, how do we decide if they evaluate the same polynomial? It turns out that there is no deterministic algorithm. But a randomized algorithm can decide (Evaluate with random input). If $p_1 = p_2$, the algorithm is always correct. If $p_1 \neq p_2$, the algorithm is correct with probability $\geq \frac{2}{3}$.

Definition: 5.1: Probabilistic Turing Machines

PTMs are TMs with 2 transition functions δ_0 and δ_1. To execute M on x, at each step, independently and randomly choose which transition function to apply. At the end, output 0 or 1. $M(x)$ is a random variable. M runs in time $T(n)$ if for any input x, M halts in $T(|x|)$ steps for all random choices. Alternatively, M has access to one extra read once tape which contains a random string.

Definition: 5.2: Bounded Probabilistic Poly-time (BPP)

BPP are decision problems efficiently solvable by PTM. $L \in$BPP if $\exists$ polytime probabilistic PTM M s.t. $\forall x \in \{0,1\}^*$, $\Pr[M(x) = L(x)] \geq \frac{2}{3}$. (The PTM M decides the instance x correctly with high probability) If $x \in L$, then $\Pr[M(x) = 1] \geq \frac{2}{3}$. If $x \notin L$, then $\Pr[M(x) = 0] \geq \frac{2}{3}$.
Equivalently, $L \in$BPP if $\exists$ poly time TM M and polynomial $p : \mathbb{N} \to \mathbb{N}$ s.t. $x \in \{0,1\}^*$, $\Pr_{r \in_R \{0,1\}^{p(|x|)}}[M(x,r) = L(x)] \geq \frac{2}{3}$, where r is a randomly sampled string of poly size as a certificate.

Definition: 5.3: Randomized Poly-time (RP)

If $x \in L$, then $\Pr[M(x) = 1] \geq \frac{2}{3}$. If $x \notin L$, then $\Pr[M(x) = 0] = 1$.
co-RP: If $x \in L$, then $\Pr[M(x) = 1] = 1$. If $x \notin L$, then $\Pr[M(x) = 0] \geq \frac{2}{3}$.

P,BPP are closed under complement. RP is not closed under complement. P$\subset$RP,co-RP$\subset$BPP.

Remark 1. Conjecture: P=BPP, likely BPP$\subset$NP, BPP$\subset$EXP, BPP$\subsetneq$NEXP

Theorem: 5.1:

RP$\subset$NP

Proof. If $x \notin L$, then the TM always reject. If $x \in L$, then the TM accepts with high probability. With the same verifier of NP, we can decide RP. □

Theorem: 5.2: Chernoff Bounds

Let $X_1, X_2, ..., X_n$ be mutually independent random variables over $\{0,1\}$, $\mu = \frac{1}{n}\sum_{i=1}^n \mathbb{E}[X_i] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n X_i\right]$, $\Pr\left[\left|\frac{1}{n}\sum X_i - \mu\right| \geq \epsilon\mu\right] \leq 2e^{-\frac{\epsilon^2\mu}{4}n}$

Theorem: 5.3: Error Reduction Theorem

$L \subset \{0,1\}^*$. Suppose there exists polytime PTM M s.t. $\forall x$, $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then $\forall$ constants $d > 0$, there exists polytime PTM M' s.t. $\forall x$, $\Pr[M'(x) = L(x)] \geq 1 - 2^{-|x|^d}$.

Theorem: 5.4: Adelman

BPP$\subset$P/poly

Proof. Assume $\Pr[M(x,r) \neq L(x)] \leq \frac{1}{2^{n+1}}$ by error reduction theorem.
Let $L \in$BPP, $x \in L$. We say a string r is bad for x if $M(x,r) \neq L(x)$.
Fraction of bad string for fixed $x \leq \frac{1}{2^{n+1}}$, so number of bad strings for $x \leq \frac{1}{2^{n+1}} 2^m$.
Fraction of bad strings for some x of length $n \leq \frac{1}{2}$.
Therefore, there exists a string r that is good for all $x \in \{0,1\}^n$. Hardwire such a string r_0 to obtain a circuit C_n s.t. on input x, $C_n(x) = M(x, r_0)$. Then $C_n(x) = L(x)$, $\forall x \in \{0,1\}^n$. □

Theorem: 5.5: Impagliazzo-Wigderson

If there exists $L \in \text{DTIME}(2^{\mathcal{O}(n)})$ that needs circuit of size $2^{\epsilon n}$, then BPP=P.

Theorem: 5.6: Sipser-Gacs-Lautemann

BPP$\subset \Sigma_2^p \cap \Pi_2^p$

Proof. We show BPP$\subset \Sigma_2^p$
Rewrite definition of BPP: $x \in L \Rightarrow \Pr[M(x,r) = \text{ accept}] \geq 1 - 2^{-n}$, and $x \notin L \Rightarrow \Pr[M(x,r) = \text{ accept}] \leq 2^{-n}$
For $x \in \{0,1\}^n$, let S_x be the set of x s.t. $M(x,r) = 1$, $|S_x| \geq (1 - 2^{-n})2^m$ or $|S_x| \leq 2^{-n}2^m$.
Let $S \subset \{0,1\}^m$, $u \in \{0,1\}^m$, define $S + u = \{x + u : x \in S\}$.
Let $k = \frac{2m}{n}$.

Claim 1: If $S \subset \{0,1\}^n$, $|S| \leq 2^{m-n}$, then for any choice of k vectors, $u_1, .., u_k$, $\bigcup_{i=1}^k (S + u_i) \neq \{0,1\}^m$

Proof. $|S + u_i| \leq 2^{m-n}$, $\left|\bigcup_{i=1}^k (S + u_i)\right| \leq \frac{2m}{n} 2^{m-n} < 2^m$. Thus they cannot cover $\{0,1\}^m$ □

Claim 2: If $|S| \geq (1 - 2^{-n})2^m$, then $\exists u_1, ..., u_k$, $\bigcup_{i=1}^k (S + u_i) = \{0,1\}^m$

Proof. Use the probabilistic method and show that if $u_1, ..., u_k$ are chosen uniformly and independently, then $\Pr[\bigcup_i (S + u_i) = \{0,1\}^m] > 0$.
For fixed $r \in \{0,1\}^m$, let B_r be the event that $r \notin \bigcup_i (S + u_i)$. We show that $\Pr[B_r] < 2^{-m}$
Let B_r^i be the event $r \notin S + u_i$, $\Pr[B_r^i] \leq 2^{-n}$, since $|S| \geq (1 - 2^{-n})2^m$.
Therefore, $\Pr[B_r] = (2^{-n})^k = 2^{-2m} < 2^{-m}$. □

With claim 1 and claim 2, $x \in L \Leftrightarrow \exists u_1, u_2, ..., u_k \in \{0,1\}^m$ s.t. $\forall r \in \{0,1\}^m$, $r \in \bigcup_{i=1}^k (S_x + u_i) \Leftrightarrow \exists u_1, ..., u_k, \forall r \in \{0,1\}^m, \vee_{i=1}^k M(x, r + u_i)$ accepts. Thus $L \in \Sigma_2^p$.

Because BPP is closed under complement, BPP$\subset \Pi_2^p$. Thus BPP$\subset \Sigma_2^p \cap \Pi_2^p$. □

Communication complexity:

Suppose A holds a string $x \in \{0,1\}^n$, B holds a string $y \in \{0,1\}^n$. A, B want to compute $f(x,y)$ with minimum communication. E.g. to deterministically determine if $x = y$, need $\mathcal{O}(n)$ communications. With randomness, only $\mathcal{O}(\log n)$ is required. With a pre-defined error correcting code, $E(x) = x'$, $E(y) = y'$. If $x = y$, then $x' = y'$. Otherwise x' and y' differ in more places. A then sends i, x'_i, B computes x'_i and y'_i

6 Interactive Proof

Definition: 6.1: Proof System

Let L be a language in which strings represent true assertions. A proof system for L is given by a verification algorithm V with the following properties:

1. Completeness: true assertions have proofs. If $x \in L$, then there exists a proof s.t. $V(x, \text{proof}) =$accept.
2. Soundness: false assertions have no proofs. If $x \in L$, then there exists a proof* s.t. $V(x, \text{proof}^*) =$reject.
3. Efficiency: $V(x, \text{proof})$ runs in time poly($|x|$)

If all three properties are satisfied, then L=NP. We can augment V with randomness and interaction to make it more powerful.

Definition: 6.2: Graph (non-)Isomorphism

The Graph Isomorphism problem $\{\langle G, H\rangle : G \cong H\} \in$NP. It always has a proof.
The Graph non-Isomorphism problem $\{\langle G, H\rangle : G \ncong H\} \in$co-NP. We don't know if it is NP. It is hard to find a proof.

Definition: 6.3: Class IP

$L \in \text{IP}[k]$ if $\exists$ probabilistic polytime TM V that has k rounds of interaction with prover $P : \{0,1\}^* \to \{0,1\}^*$. To verify x, follow Fig. 1. Verifier is polytime, prover can answer any string, WLOG $|q_i|, |a_i| \leq \text{poly}(n)$.
Let Π denote the set of communicated strings. In the final round $V = 0, 1$. V is complete. *i.e.* if $x \in L$, $\exists P$ s.t. $\Pr[V(x, \Pi) = 1] \geq \frac{2}{3}$; if $x \notin L$, $\forall P^*$, $\Pr[V(x, \Pi) = 1] \leq \frac{1}{3}$. The probability is over random coin tosses of V.
IP$= \bigcup_c \text{IP}[n^c]$.

Figure 1: Verfication Procedure

$$V \underset{a_1 = g(x, q_1)}{\overset{q_1 = f(x)}{\rightleftarrows}} P$$

$$\underset{a_2 = g(x, q_1, a_1, q_2)}{\overset{q_2 = f(x, q_1, a_1)}{\rightleftarrows}}$$

Observations:

1. NP$\subset$IP: verifier sends empty question, prover sends the certificate, then verifier decides.
2. BPP$\subset$IP
3. Just like BPP, the error can be exponentially reduced. ($\frac{2}{3} \to 1 - 2^{-n^c}$, $\frac{1}{3} \to 2^{-n^c}$)
4. IP$\subset$PSPACE: There is no advantage making prover to be out of PSPACE. The answeers can be found in PSPACE always. Need to consider the best answer for multiple rounds without knowing what question is coming, but the best answer is in PSPACE.

5. public coin v.s. private coin: in private coin model, prover doesn't see the coin tosses of verifier. In public coin model, verifier tosses the coin and send the coin toss to prover. The prover then determines the questions and answers based on the coin toss. The results are deterministic.

6. WLOG, the prover can be deterministic

7. IP with deterministic verifier is NP

***Theorem:* 6.1: Shamin**

IP=PSPACE (if a language is in PSPACE, it can be verified by a poly time verifier with interaction)

***Theorem:* 6.2:**

GNI= $\{(G_1, G_2) : G_1 \ncong G_2\}$. GNI$\in$IP.

Proof. Verifier V toss random coin, pick one of G_1, G_2, randomly permute vertices to produce H. V then sends H to P.
If G_1, G_2 are not isomorphic, P can answer if $H \cong G_1$ or $H \cong G_2$ correctly. If $G_1 \cong G_2$, prover has no idea, the best result will be random guesses. □

***Theorem:* 6.3: Auther-Merlin**

IP$[k]$ = AM$[k + 2]$.

***Theorem:* 6.4:**

co-NP$\subset$IP

Proof. Let E#SAT $= \{(\phi, k) : \phi$ has exactly k satisfying assignments$\}$
We show that E#SAT$\in$IP.
Observations: if $\phi(x_1, ..., x_n)$ has exactly k satisfying assignments, then $\exists k_0, k_1$ s.t.

1. $k_0 + k_1 = k$

2. $\phi_0(x_2, ..., x_n) = \phi(0, x_2, ..., x_n)$ has k_0 satisfying assignments.

3. $\phi_1(x_2, ..., x_n) = \phi(1, x_2, ..., x_n)$ has k_1 satisfying assignments.

Idea: both V and P knows ϕ and k. Prover sends k_0, k_1, verifier checks that $k_0 + k_1 = k$, randomly select $b \in \{0, 1\}$ for first variable. Prover recursively show ϕ_b has k_b satisfying assignments and reduce the number of variables.
If ϕ has k satisfying assignments, then V accepts with probability 1.
If ϕ does not have k satisfying assignments, V rejects with probability $\geq \frac{1}{2^n}$. (always select the "true" b, the lie will be caught at the end with single variable)
This protocol is too weak and requires exponentially many rounds to reject.

To improve the performance, the key idea is to use arithmetization.
Allow variables to take values in a large field $\mathbb{F}$ s.t. $\{0, 1\} \subset \mathbb{F}$.
Extend the formula to a more robust function $\tilde{\phi} : \mathbb{F}^n \to \mathbb{F}$ s.t. $\tilde{\phi}|_{\{0,1\}^n} = \phi$.
We want to make sure that if the prover cheats on one value, the prover has to cheat on more values. We extend ϕ to multivariate low degree polynomial in $\mathbb{F}$.
Robustness property: two distinct low degree polynomials cannot take the same value in many (more than degree) locations.

$\phi(x_1, ..., x_n) \to \tilde{\phi}(\tilde{x_1}, ..., \tilde{x_n})$ s.t. $\tilde{x_i} = x_i$, $\neg\phi \to 1 - \tilde{\phi}$, $\phi \tilde{\wedge} \psi \to \tilde{\phi}\tilde{\psi}$, $\phi \tilde{\vee} \psi \to 1 - (1 - \tilde{\phi})(1 - \tilde{\psi})$
Each clause becomes a degree 3 polynomial. With m clauses, we get at most degree $3m$

$$\sum_{x_1 \in \{0,1\}, ..., x_n \in \{0,1\}} \tilde{\phi}(x_1, ..., x_n) = k$$

The equation holds if and only if ϕ has k satisfying assignments.

Side note: Primes= $\{x : x \text{ is a prime}\} \in$P, so we can decide if a number 2^n is a prime in poly(n).

Formal protocal: input $\phi(x_1, ..., x_n)$ and an integer k.

1. P, V compute $\tilde{\phi}(x_1, ..., x_n)$ arithmetization and a finite field $\mathbb{F}$ s.t. char($\mathbb{F}$) $> 2^d$ where $d = |\phi|$ so $\tilde{\phi}$ has degree $\leq d$
2. Let $p_1(x) = \sum_{x_2, ..., x_n} \tilde{\phi}(x, x_2, ..., x_n)$ be a univariate polynomial. P computes p_1 and sends it to V.
3. V checks if $p_1(0) + p_1(1) = k$. If not, reject. Otherwise, choose α_1 uniformly from $\mathbb{F}$ and send to P. $p_1(\alpha_1) = \sum_{x_2, ..., x_n} \tilde{\phi}(\alpha_1, x_2, ..., x_n)$. Then prover compuites $p_2(x) = \sum_{x_3, ..., x_n} \tilde{\phi}(\alpha_1, x, ..., x_n)$
4. For $i = 2, ..., n$, P sends $p_i(x) = \sum_{x_{i+1}, ..., x_n} \tilde{\phi}(\alpha_1, ..., \alpha_i, x_{i+1}, ..., x_n)$ to V. V checks if $p_i(0) + p_i(1) = p_{i-1}(\alpha_{i-1})$. If not, reject. Otherwise, sample α_i uniformly from $\mathbb{F}$ and send to P. Now the prover should show $p_i(\alpha_i) = \sum_{x_{i+1}, ..., x_n} \tilde{\phi}(\alpha_1, ..., \alpha_i, x_{i+1}, ..., x_n)$.
5. Repeat for n rounds. The last polynomial can be esily computed by V. V accepts if $p_n(\alpha_n) = \tilde{\phi}(\alpha_1, ..., \alpha_n)$.

Efficiency: arithmetization, each check, sampling etc are poly time.
Completeness: An honest prover can pass the tests.
Soundness: If ϕ does not have k satisfying assignments, then no matter what poly p_i^* is sent, V accepts with probability $\leq \frac{nd}{|\mathbb{F}|} \leq \frac{d^2}{|\mathbb{F}|} \leq \frac{1}{3}$.
If ϕ does not have k satisfying assignments, then either $p_1^*(0) + p_1^*(1) \neq k$ or $p_1^* \neq p_1$.
If $p_1^* \neq p_1$ with probability $\geq 1 - \frac{d}{|\mathbb{F}|}$, $p_1^*(\alpha_1) = p_1(\alpha_1)$. After setting first variable, P is left with false assertion to prove.
Later rounds are similar. If $p_{i-1}^*(\alpha_{i-1}) \neq p_{i-1}(\alpha_{i-1})$, no matter what p_i^* is sent by the prover, either $p_i^*(0) + p_i^*(1) \neq p_{i-1}^*(\alpha_{i-1})$, or $p_i^* \neq p_i$. P is left with false assertions to prove. □

7 Probabilistically Checkable Proofs (PCP)

***Definition:* 7.1: PCP Verifier**

Assume the verifier V has random access to Π, and can query any bit of Π using address tape.
Let L be a language, $q, r : \mathbb{N} \to \mathbb{N}$, L has a $(r(n), q(n))$ PCP verifier if there exists a probabilistic poly-time verifier V with the following properties:

1. Efficiency: on input $x \in \{0,1\}^n$ and given access to $\Pi \in \{0,1\}^*$, V uses $\leq r(n)$ random coins and makes at most $q(n)$ nonadaptive queries to location of Π of output 0/1. $V^{\Pi}(x)$ is a random variable.
2. Completeness: if $x \in L$, there exists $\Pi \in \{0,1\}^*$ s.t. $\Pr[V^{\Pi}(x) = 1] = 1$
3. Soundness: if $x \notin L$, $\forall \Pi^* \in \{0,1\}^*$, $\Pr[V^{\Pi^*}(x) = 1] \leq \frac{1}{2}$.

WLOG, Π has length at most $2^{r(n)}q(n)$

***Definition:* 7.2: PCP Language**

$L \in \text{PCP}(r(n), q(n))$ if there exists constant $c, d > 0$ s.t. L has a $(cr(n), dq(n))$ PCP verifier.

***Theorem:* 7.1: PCP Theorem (ALMSS 1992)**

$\text{NP} = \text{PCP}(0, \text{poly}(n)) = \text{PCP}(\log n, 1)$

Observations:

1. The constant $\frac{1}{2}$ can be boosted.
2. $\text{PCP}(r(n), q(n)) \subset \text{NTIME}(2^{\mathcal{O}(r(n))}q(n))$, can toss $2^{\mathcal{O}(n)}$ coins to explore all locations, so $\text{PCP}(\log n, 1) \subset \text{NTIME}(2^{\mathcal{O}(\log n)})$ =NP is trivial.

7.1 Hardness of Approximation

***Definition:* 7.3: MAX-3SAT**

Given a 3-CNF formula ϕ, find an assignment such that maximizes number of satisfied clauses. MAX-3SAT is NP-hard.

An algorithm A is $\rho-$approximation ($\rho \leq 1$) for MAX-3SAT if $\forall$ 3-CNF formula ϕ with m clauses, $A(\phi)$ outputs assignment that satisfies $\geq \rho \text{val}(\phi) m$ clauses, where $\text{val}(\phi)$ is max function of satisfiable clauses.

Could it be for $\rho = 1 - \epsilon$, $\forall \epsilon > 0$, there exists a $(1-\epsilon)-$approximation algorithm for MAX-3SAT?
If P$\neq$NP, then there is no such algorithm.

***Theorem:* 7.2:**

It is NP hard to get $\rho-$approximation for all $\rho < 1$. But there exists $\rho < 1$ s.t. $\forall L \in$NP, there is a polytime function $f : \{0,1\}^* \to \{0,1\}^*$. If $x \in L$, then $\text{val}(f(x)) = 1$. If $x \notin L$, then $\text{val}(f(x)) \leq \rho$.

***Theorem:* 7.3: $\frac{7}{8}-$approximation (Hastad)**

If there is a $\frac{7}{8} + \epsilon-$approximation for MAX-3SAT for any $\epsilon > 0$, then P=NP. There exists a $\frac{7}{8}-$approximation algorithm with semidefinite programming.

Definition: 7.4: Constrained Satisfiability Problem

In $q-$CSP, an instance ϕ is a collection of functions $\phi_1, ..., \phi_m : \{0,1\}^n \to \{0,1\}$ (m constraints). Each ϕ_i depends on q input locations.
val(ϕ) =max over all $u \in \{0,1\}^*$ of functions of $\phi_1, ..., \phi_m$ that are satisfied.

Definition: 7.5: Gap CSP

Given $q \in \mathbb{N}$, $\rho < 1$, $\rho-$GAP$q-$CSP problem determines

1. val(ϕ) $= 1$
2. val(ϕ) $< \rho$.

It is NP-hard. $\forall L \in$NP, there exists a polytime $f : \{0,1\}^* \to \{0,1\}^*$ where the codomain is $q-$CSP. If $x \in L$, val$(f(x)) = 1$. If $x \notin L$, val$(f(x)) < \rho$.

Theorem: 7.4:

$\exists q, \rho < 1$ s.t. $\rho-$GAP$q-$CSP is NP-hard.

Note that Therorem 7.1, 7.2, 7.4 are equivalent

Proof. (Sketch)
7.2⇒7.4, Apply to multiple instances
7.2⇐7.4, represent ϕ_i by 3-CNF, transform into a 3SAT instance

7.4⇐7.1, q is the same for both.
On input x, V computes $f(x)$ to get some $q-$CSP instance $\phi = \{\phi_i\}_{i=1}^m$.
Expects Π to be assignments to variables. Pick $i \in \{1, ..., m\}$. Check ϕ_i is satisfied. □

8 Cryptography

Let x be the plain text Alice sends to Bob. A computationally bounded adversary Eve can get any information from the channel between Alice and Bob. Let $y = E(x)$ be the encoded text. It should be impossible for Eve to guess x from y, but Bob can figure out x from y.

***Definition:* 8.1: Private Key Perfect Secrecy**

A&B share a secret key k (a string chosen at random). A sends $y = E_k(x)$ to B. B computes $x = D_k(E_k(x))$.
A (E, D)-scheme is perfect secrecy for keys u_n of length n, if $\forall x, x' \in \{0,1\}^m$, $m \le n$, distributions of $E_{u_n}(x)$ and $E_{u_n}(x')$ are identical. If $m > n$, perfect secrecy is impossible.

***Definition:* 8.2: One Time Pad**

Assume $m = n$, $x \in \{0,1\}^m$, $k \leftarrow \{0,1\}^m$, $E_k(x)$ =bitwise XOR (addition mod 2) of k and x. $D_k(E_k(x)) = x$ by XOR k and $E_k(x)$.

If we send more than one message using the same key, Eve can learn information from the messages.

***Theorem:* 8.1:**

If P=NP, and (E, D) is an encryption/decryption scheme (polytime computable) with key length $n <$ message length m, then there exists a polytime algorithm $\bar{A}$ s.t. $\exists x_0, x_1 \in \{0,1\}^m$, $\Pr_{k,b \leftarrow \{0,1\}^n}\left[\bar{A}(E_k(x_b)) = 1\right] \ge \frac{3}{4}$.

Proof. Let $x_0 = 0^m$, $s = \text{supp}\{E_{u_n}(0^m)\}$.
Since P=NP, membership in S can be verified in polytime.
Decision problem: is $y \in S$? Certificate: the secret key u_n.
If $\exists x$ s.t. $\Pr_{k \leftarrow \{0,1\}^n}[E_k(x) \in S] \le \frac{1}{3}$, then $x_1 = x$, $\bar{A}$ on input y will just check if $y \in S$. If $y \in S$, $\bar{A}$ output 0, else output 1.
Assume $\forall x \in \{0,1\}^m$, $\Pr[E_{u_n}(x) \in S] > \frac{1}{2}$.
Create a bipartite graph from $\{0,1\}^m$ to $\{0,1\}^n$. If $E_{u_n}(x) \in S$, color the edge red.
There exists a key $k \in \{0,1\}^n$ with $> 2^{m-1}$ red edges, since $\Pr[E_{u_n}(x) \in S] > \frac{1}{2}$.
But $|S| \le 2^n \le 2^{m-1}$. Therefore, there exists key k and x', x'' s.t. $E_k(x') = E_k(x'')$.
However, this is not a valid (E, D)-scheme, because B cannot decrypt $E_k(x')$ and $E_k(x'')$. ☐

***Definition:* 8.3: One-way Function**

A poly-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function if for all probabilistic poly-time algorithm A, $\Pr_{x \leftarrow \{0,1\}^n}[A(y) = x' \text{ s.t. } f(x') = y] \le \epsilon(n)$ for $\epsilon(n) = \frac{1}{n^{\omega(1)}}$.

If one-way function exists, then P≠NP.

***Definition:* 8.4: Secure Pseudorandom Generator**

$G : \{0,1\}^* \to \{0,1\}^*$ is a polytime computable function, $l : \mathbb{N} \to \mathbb{N}$ s.t. $l(n) > n$, G is a secure PRG of strech $l(n)$ if $|G(x)| = l(|x|)$ for all $x \in \{0,1\}^n$, and for all probabilistic polytime algorithm A, $\Pr[A(G(u_n)) = 1] - \Pr\left[A(U_{l(n)}) = 1\right] \le \epsilon(n)$. *i.e.* A cannot tell the pseudo-distribution $G(u_n)$ from true random distribution $U_{l(n)}$.

One-way functions are enough to construct PRGs.

***Theorem:* 8.2: Goldreich-Levin**

Let $\{f_n\}$ be a family of one-way permutations. Let $x \leftarrow \{0,1\}^n$, $r \leftarrow \{0,1\}^n$, $g(x,r) = (f(x), r, \langle x, r \rangle)$, where $\langle x, r \rangle$ is the inner product, $g : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ is not a random distribution, but it is random for a computationally bounded algorithm. For all probabilistic poly-time algorithm A, $\Pr_{x,r}[A(f(x), r) = \langle x, r \rangle] \leq \frac{1}{2} + \epsilon(n)$.