Computational Models Cheat Sheet

Saar hagever - TAU 2022

Algorithm Definition

- Finite set of available instructions
- Made of finite instructions
- Discrete steps taking finite time
- Each step depends only on the previous ones
- No limit to steps or space during execution

Turing Machine

A Turing Machine M is (Q, Σ, δ, q_0)

$Q(\ni q_i), h \notin Q$	Set of states, h halting state	
$\Sigma(\ni \sigma, \sigma', \dots)$	Set of symbols	
$q_0 \in Q$	Initial state	Σ^* is the free monoid
$\delta \subseteq (Q \times \Sigma) \times (Q \cup \{h\} \times \Sigma \times \{L, R, -\})$	is the transition function	
if $(q, \triangleright, q', \sigma, D) \in \delta$	then $\sigma = \triangleright, D = R$	

generated from Σ. " $\cdot$ " is the associative string concatenation operator. ϵ is the empty string and identity element.

Configuration

A configuration C of a TM M is a quadruple: $(q, u, \sigma, v) \in (Q \cup \{h\}) \times \Sigma^* \times \Sigma \times \Sigma^F)$ where $\Sigma^F = \Sigma^* \cdot (\Sigma \backslash \{\#\}) \cup \{\epsilon\}$.
σ is the currently selected symbol under the head of the machine. Short notation is $(q, u\underline{\sigma}v)$ or (q, w) when position is not needed.

Computations

A computation is a finite succession of computation steps

$(q, w) \to (q', w')$	Computation step
$(q_0, w) \to^* (q', w')$	Reflexive and transitive closure of comp. steps.
$(q_0, w) \to_M^n (q', w')$	Machine M computes w' in n steps.
$\downarrow, \uparrow$	Converges and diverges

T(uring)-Computability

Let $\Sigma, \Sigma_0, \Sigma_1$ be alphabets, let $\#, \triangleright \notin \Sigma_0 \cup \Sigma_1$ and $\Sigma_0 \cup \Sigma_1 \subset \Sigma$.
Let f be a function $f : \Sigma_0^* \to \Sigma_1^*$.

$$f \text{ is turing computable by a machine } M \Leftrightarrow \forall w \in \Sigma_0^* \ : f(w) = z \Leftrightarrow (q_0, \underline{\triangleright} w) \to_M^* (h, \triangleright z \underline{\#})$$

sectionProblems and Functions

Poly reduction

$L_1 \leq_P L_2$ Then exsit $f \in O(n) \mapsto |f(w)| = n^k$ for some k

Total Function

$f : A \to B$, subset of $A \times B$ is total $\Leftrightarrow$ $\forall a \in A \ \exists b \in B \ : (a, b) \in f$ f is defined everywhere
$(a, b), (a, c) \in f \Rightarrow b = c$ uniqueness

Partial Function

$f : A \to B$, subset of $A \times B$ is a *partial* function $\Leftrightarrow$
$(a, b), (a, c) \in f \Rightarrow b = c$ uniqueness
$\exists$ at least a $b \in B \ : \ f(a) = b$ not required to be defined everywhere

Theorem. *Computable functions are $\#(\mathbb{N})$. Total computable functions are $\#(\mathbb{N})$. There exists functions that are not computable.*

Proof. Computable functions can be given a Gödel numbering. Therefore there exists a 1-1 mapping with natural numbers. All the possible total functions $\{f : \mathbb{N} \to \mathbb{N}\}$ are $2^{\#(\mathbb{N})}$. □

Theorem. ***Padding Lemma**: every computable function φ_i has $\#(\mathbb{N})$ indexes. $\forall i. \exists A_i$ infinite set of indexes such that $\forall y \in A_i. \varphi_y = \varphi_i$*

Proof. If a program P_j in WHILE computes φ_i, one can build $\#(\mathbb{N})$ other programs that compute the same thing, semantically equal to φ_i by just adding a `skip` command at the end of the program. The new program's index will change because an effective numbering depends only on the syntactic description of the program. □

Theorem. *The class of Primitive Recursive functions (and extensions) does not contain all the effectively computable functions:*

Proof. reductio ad absurdum
1: Encode and enumerate p.r. functions $f_0, \dots, f_n$. There are $\mathbb{N}$
2: Define diagonal func. $g(x) = f_x(x) + 1$ Intuitively computable
3: $\forall n. g(n) \neq f_n(n) \Rightarrow \forall n. g \neq f_n$ g doesn't appear in list
Implies there is no formalism for expressing only and all total functions. □

Recursion Axiom

Let h be a p.r. func. in $k+1$ variables, g be a p.r. func in $k-1$ variables. The function f in k variables defined as follow is then p.r.

$$f = \begin{cases} f(0, x_2, \dots, x_k) & = & g(x_2, \dots, x_k) & \text{final step} \\ f(x_1 + 1, \dots, x_k) & = & h(x_1, f(x_1, \dots, x_k), x_2, \dots, x_k) & \text{iteration} \end{cases}$$

This represents a **FOR** loop. x_1, the first variable of f is the decrementing counter. g is the final step. h represents loop steps. This is guaranteed to terminate. One can define basic maths primitives with p.r. functions.

Theorem. ***Enumeration**: $\exists z. \forall i, x. \ \varphi_z(i, x) = \varphi_i(x)$*

Proof. Guarantees that a formalism that expresses all computable function has an universal algorithm. Therefore there exists an *Universal Turing Machine.*
$\varphi_z(i, x) = U(\mu y. T(i, x, y)) = \varphi_i(x)$ □

Turing Machines

Deterministic TMs are seen in the *Computability Theory Cheat Sheet.*

K-tapes Turing Machines

Let $k \geq 1$ be the number of tapes. A k-tape TM is a quadruple (Q, Σ, δ, q_0). Special symbols $\#, \triangleright, \in \Sigma$, while $L, R, - \notin \Sigma$. Since we are only considering decision problems, YES, NO $\notin Q$ are the halting states. The transition function differs but is subject to the same restrictions as a classical TM.

$$\delta : Q \times \Sigma^k \to Q \cup \{\text{YES}, \text{NO}\} \times (\Sigma \times \{L, R, -\})^k$$

IO Turing Machines

Are useful to study space complexity. Any k-tape TM $M = (Q, \Sigma, \delta, q_0)$ is of IO type $\Leftrightarrow$ δ is subject to the following constraints. Consider $\delta(q_1, \sigma_1, \dots, \sigma_k) = (q', (\sigma_1', D_1), \dots, (\sigma_k', D_k))$

$\sigma_1' = \sigma_1$	First tape is read-only
$D_k = R$ or $(D_k = -) \Rightarrow \sigma_k' = \sigma_k$	Last tape is write-only
$\sigma_1 = \# \Rightarrow D_1 \in \{L, -\}$	Input tape is right-bounded

Nondeterministic Turing Machines

A **nondeterministic** TM is a quadruple $N = (Q, \Sigma, \Delta, q_0)$ where Q, Σ, q_0 are the same as in other turing machines. They can be of IO and k-tape type. The difference rilies in the fact that

$$\Delta \subseteq (Q \times \Sigma) \times ((Q \cup \{\text{YES}, \text{NO}\}) \times \Sigma \times \{L, R, -\})$$

Is not a transition function but a **transition relation**. The same syntax as for other TMs is used for nondet. TMs configurations and computations. A nondeterministic TM N decides I if and only if $x \in I \Leftrightarrow$ there exists **at least** a computation such that $(q_0, \underline{\triangleright}, x, \triangleright, \dots, \triangleright) \to_N^* (\text{YES}, w_1, \dots, w_k)$

Note. A single computation on a nondeterministic TMs is not a tree. Many computations together, can be arranged in a tree.

The **degree** d of nondeterminism of a NDTM $N = (Q, \Sigma, \Delta, q_0)$ can now be defined as $d = \max\{\deg(q, \sigma) \mid q \in Q, \sigma \in \Sigma\}$, where $\deg(q, \sigma) = \#\{(q', \sigma', D) \mid ((q, \sigma), (q', \sigma', D)) \in \Delta\}$

Reductions

A (decision) problem (a set) A reduces to B with *reduction* f, in symbols $A \leqslant_f B$ when $a \in A \Leftrightarrow f(a) \in B$. We also have that $A \leqslant_f B \Leftrightarrow \overline{A} \leqslant_f \overline{B}$ because $x \in \overline{A} \Leftrightarrow x \notin A \Leftrightarrow f(x) \notin B \Leftrightarrow f(x) \in \overline{B}$.

Reduction Relations

A relation of reductions ($\leqslant_F$) can be defined on a class of functions F as follows, note that $\leqslant_F$ is also a partial pre-order.

$$A \leqslant_F B \Leftrightarrow \exists f \in F. \ A \leqslant_f B$$

Properties

Let $\mathcal{D}$ and $\mathcal{E}$ two classes of problems such that $\mathcal{D} \subseteq \mathcal{E}$, and $\mathcal{D} \subseteq \mathcal{H}$. A reduction relation $\leqslant_F$ classifies $\mathcal{D}$ and $\mathcal{E} \Leftrightarrow \forall A, B, C$ (problems):

1: Reflexivity	$A \leqslant_F A$
2: Transitivity	$A \leqslant_F B, B \leqslant_F C \Rightarrow A \leqslant_F C$
3: $\mathcal{D}$ closed by reduction	$A \leqslant_F B, B \in \mathcal{D} \Rightarrow A \in \mathcal{D}$
4: $\mathcal{E}$ closed by reduction	$A \leqslant_F B, B \in \mathcal{E} \Rightarrow A \in \mathcal{E}$

Equivalently:

1: Identity	$\text{id} \in F$
2: Closed by composition	$f, g \in F \Rightarrow f \circ g \in F$
3:	$f \in F, B \in \mathcal{D} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{D}$
4:	$f \in F, B \in \mathcal{E} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{E}$

Hard and Complete Problems

If $\leqslant_F$ classifies $\mathcal{D}$ and $\mathcal{E}$, then $\forall$ problems A, B, H.

- $A \leqslant_F B \wedge B \leqslant_F A \Rightarrow A \equiv B$
- H is $\leqslant_F$-hard for $\mathcal{E}$ if $\forall A \in \mathcal{E}. \ A \leqslant_F H$
- H is $\leqslant_F$-complete for $\mathcal{E}$ if $\forall A \in \mathcal{E}. \ A \leqslant_F H \wedge H \in \mathcal{E}$

Theorem. *If $\leqslant_F$ classifies $\mathcal{D}$ and $\mathcal{E}$, $\mathcal{D} \subseteq \mathcal{E}$ and C is complete for $\mathcal{E}$ then $C \in \mathcal{D} \Leftrightarrow \mathcal{D} = \mathcal{E}$*

Proof. ($\Leftarrow$) is obvious. ($\Rightarrow$): Let $C \in D$ and $A \in \mathcal{E}$. By completeness, $A \leqslant_F C$ and $A \in \mathcal{D}$, then $\mathcal{E} \subseteq \mathcal{D}$ which implies $\mathcal{E} = \mathcal{D}$ □

Theorem. ***Completeness is "transitive" for reductions**: If A is complete for $\mathcal{E}$, $A \leqslant_F B$ and $B \in \mathcal{E}$ then B is complete for $\mathcal{E}$.*

Boolean Circuit

Definition

A Boolean circuit C "computes" a (finite) function $f : \{0,1\}^n \mapsto \{0,1\}^m$ over De Morgan basis
A **circuit ensembles** is an infinite family of circuits $\mathcal{A} = \{A_n\}_{n \in N}$. A_n has n input bits, $\mathcal{A}$ has m bit output, if each A_n has $m(n)$ output bits (i.e., m is a function from N to N, might be constant, e.g., m(n) = 1). Circuit ensemble can handle any input length, i.e., compute any (even infinite) functions, but the description size is infinite!.

Theorem. *(Shanon) Every $L \in \{0,1\}^n$ can be decided by a (De Morgan) circuit of size $O(\frac{2^n}{n})$*

Regular language

Lemma. *(pumping lemma): for every regular language $\mathcal{L}$, there exists $\ell > 0$ (a.k.a. pumping length) s.t. every $s \in \mathcal{L}$ with $|s| \geq \ell$ can be written as $s = xyz$ with with $\forall i \geq 0 x y^i z \in \mathcal{L}, |y| > 0, |xy| \neq 0$*

Classical Results

Theorem. *The class of Primitive Recursive functions (and extensions) does not contain all the effectively computable functions:*

Proof. reductio ad absurdum
1: Encode and enumerate p.r. functions $f_0, \dots, f_n$. There are $\mathbb{N}$
2: Define diagonal func. $g(x) = f_x(x) + 1$ Intuitively computable
3: $\forall n. g(n) \neq f_n(n) \Rightarrow \forall n. g \neq f_n$ g doesn't appear in list
Implies there is no formalism for expressing only and all total functions. □

Resulting regular expression:

$L' = \{0^i 1^j : i, j \geq 0$ and $\exists w \in L s.t |w| = i + j\}$ is regular if L regular **T**. built NFA
$Q \times \{0,1\}, \delta'((q,0),0) = \{(\delta(q,\sigma),0) : \sigma \in \Sigma^*\}.\delta'((q,0),\epsilon) = \{(\delta(q,1),\sigma) : \sigma \in \Sigma^*\}$
$L' = \{0^i 1^j : i, j \geq 0$ and $\exists w \in L s.t |w| = |i - j|\}$is regular if L regular **F**. take $L = \epsilon$

r = r1* r2(r4+r3r1*r2)* where: r1: initial → initial r2: initial → accepting r3: accepting → initial r4: accepting → accepting

Complexity Classes

$$\mathcal{P} = \bigcup_{k \geq 1} \text{TIME}(n^k) \qquad \mathcal{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k)$$

$$\text{PSPACE} = \bigcup_{k \geq 1} \text{SPACE}(n^k) \qquad \text{NPSPACE} = \bigcup_{k \geq 1} \text{NSPACE}(n^k)$$

$$\text{EXP} = \bigcup_{k \geq 1} \text{TIME}(2^{n^k}) \quad \text{LOGSPACE} = \bigcup_{k \geq 1} \text{SPACE}(k \times \log(n))$$

Hierarchy

$$\text{LOGSPACE} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subset \text{EXP} \subset R \subset RE$$

Since LOGSPACE $\subsetneq$ PSPACE, at least one of those inclusions is strict:

$$\text{LOGSPACE} \subseteq \mathcal{P}, \mathcal{P} \subseteq \mathcal{NP}, \mathcal{NP} \subseteq \text{PSPACE}$$

It is not yet know which one of those is a strict inclusion.

Theorem. LOGSPACE $\subseteq \mathcal{P}$. *Proof.* Let $I \in$ LOGSPACE. There $\exists$ a TM that computes $x \in I$ in $\mathcal{O}(\log|x|)$ space. M can range through $\mathcal{O}(|x| \log|x| \#Q \#\Sigma^{\log|x|})$ configurations. A halting computation cannot cycle on configurations. So M computes in $\mathcal{O}(|x|^k)$ for some k.

Theorem. (Savitch) NPSPACE = PSPACE.

Theorem. Hierarchy of time and space. If f is appropriate, then $\text{TIME}(f(n)) \subsetneq \text{TIME}((f(2n+1))^3)$, and $\text{SPACE}(f(n)) \subsetneq \text{SPACE}(f(n) \times \log f(n))$. **Corollary.** $\mathcal{P} \subsetneq \text{EXP}$
Proof. $\mathcal{P} \subset \text{EXP}$ is obvious because 2^n grows faster than any polynomial. It is strict because $\mathcal{P} \subseteq \text{TIME}(2^n) \subsetneq \text{TIME}((2^{(2n+1)})^3) \subseteq \text{TIME}(2^{n^2})$. This corollary, together with the fact that $\text{NTIME}(f(n)) \subseteq \text{TIME}(c^{f(n)})$ lets us conclude that $\mathcal{NP} \subseteq \text{EXP}$.

Theorem. Hierarchy 2. Let f be an appropriate measuring function, and k a constant. Then
$\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$ $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$
$\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n + f(n)})$

Resulting regular expression:

$$L_2 = \{y : \exists x, z \ \ s.t. |x| = |z| \text{ and } xyz \in L\}$$

(Example) is regular.onsider some DFA for L. For every $n \geq 0$, let A_n be the set of states s such that there is *some* word of length n which leads the DFA from the initial state to s. Let B_n be the set of states t such that there is *some* word of length n which leads the DFA from t to an accepting state. Finally, for any two states s, t, let $R_{s,t}$ be the (regular) set of words leading the DFA from s to t. We have

$$L_2 = \bigcup_{n \geq 0} \bigcup_{\substack{s \in A_n \\ t \in B_n}} R_{s,t}.$$

Since there are only finitely many possibilities for s, t, the union is in fact finite, and so regular

Theorems on Complexity of Turing Machines

Theorem. ***Reduction of tapes*** *Let M be a k-tape TM, deciding I in deterministic time f, then $\exists$ a 1 tape TM M' that decides I in time $\mathcal{O}(f^2)$*

Proof. (Only a draft): Build a 1 tape TM M' by introducing two symbols $\triangleright'$ and $\triangleleft'$ to denote the start and end of the k-th tape. Introduce $\#\Sigma$ new symbols $\overline{\sigma_i}$ to remember each tape's head location. To generate the initial configuration $(q, \triangleright\triangleright' x \triangleleft' (\triangleright'\triangleleft')^k)$, $2k + \#\Sigma$ states and $\mathcal{O}(|k|)$ steps are needed. To simulate a move of M, M' iterates the input datum from left to right, and back, 2 times: find the marked $\overline{\sigma_i}$ symbols, the second time write the new symbols. If a tape has to be extended, the $\triangleleft'$ parens have to be moved and a cascade happens. This takes $\mathcal{O}(f(|x|))$ time, for each move of M. Since M takes $\mathcal{O}(f(|x|))$ time to compute an answer, M' will take $\mathcal{O}(f(|x|)^2)$ □

Corollary. *Parallel machines are polinomially faster than sequential machines.*

A machine cannot use more space than time

Theorem. Linear speedup If $I \in \text{TIME}(f)$, then $\forall \epsilon \in \mathbb{R}^+$. $I \in \text{TIME}(\epsilon \times f(n) + n + 2)$. *Proof.* (Draft): Build M' from M solving I, compacting m symbols of M into 1 of M', in function of ϵ. The states of M' will be triples $[q, \sigma_{i_1}, \ldots, \sigma_{i_m}, k]$ meaning the TM is in state q and has cursor on k-th symbol of $\sigma_{i_1}, \ldots, \sigma_{i_m}$. M' then needs 6 steps to simulate m steps of M. Therefore, M' will take $|x| + 2 + 6 \times \lceil \frac{f(|x|)}{m} \rceil$. Then $m = \lceil \frac{6}{\epsilon} \rceil$. □

Same principle can apply to SPACE with **linear space compression**. If $I \in \text{SPACE}(f(n))$, then $\forall \epsilon \in \mathbb{R}^+$. $I \in \text{SPACE}(\epsilon \times f(n) + 2)$

Theorem. *For each k-tape TM M that decides I in deterministic time f there exists an IO TM M' with k+2 tapes that computes I in time $c \times f$ for some constant c.*

Proof. M' copies the first M tape to the second tape, in $|x| + 1$ steps. Then, M' operates identically to M. If and when M halts, M' copies the result to the tape $k + 2$, in at most $f(|x|)$ steps. M' computation was $2f(|x|) + |x| + 1$ steps long. □

Theorem. ***Exponential loss in determinization, or bruteforce*** *If $I \in NTIME(f(n))$ and is computed by k-tape N, it can also be computed by a deterministic TM M with $k + 1$ tapes in time $\mathcal{O}(c^{f(n)})$ with an exponential loss of performance. In short,* $\text{NTIME}(f(n)) \subseteq \text{TIME}(c^{f(n)})$

Proof. Let d be the degree of nondeterminism of N. $\forall q \in Q, \sigma \in \Sigma$ sort $\Delta(q, \sigma)$ lexicographically. Every computation of length t carried by N is a sequence of choices that can be seen as a sequence of natural numbers $(c_1, \ldots, c_t)$ with $c_i \in [0..d-1]$. The det. TM M considers these successions in an increasing order, visiting the tree of nondeterministic computations, one at a time. Therefore $M(x) \downarrow \Leftrightarrow N(x) \downarrow$, also, all the possible simulations can be $\mathcal{O}(d^{f(n)+1})$. □

Space complexity

$$L = DSPACE(\log(n)) \qquad NL = NSPACE(\log(n))$$

$$\text{PSPACE} = \bigcup_{k \geq 1} \text{SPACE}(n^k) \qquad co - NL = NL \qquad DTIME(t(n)) \subseteq DSPACE(t(n))$$

STCON,PATH is NL-complete under log reductions.

NL clousre If the language A is NL-complete then
1. $A \in NL$ 2. Each problem in NL is logspace reducible to A
Since $coNL = NL$ each problem in $coNL$ is logspace reducible to A as well. Thus A is $coNL$-complete.

Any $L \in \mathcal{NP}$ have Log Verifier for L is D-TM with 3 types $(w, c,$working type$)$ V run in **Poly** time $|x|$ and using $\log(|x|)$ types from working type. if $x \in L$ exsit w s.t V accept . if $x \notin L$ for all w V reject. $L \in NL \Leftrightarrow$ exsit Right only log time verifier.

Composition of space-bounded functions

is a bit tricky: Thm: if f and g are computable in space $s_f(n).g_f(n)$. respectively, then gf is computable in space (with m_f being a bound on the output length of f).

$$s_f(n) + \log(m_f(n)) + \log(m_f(n))$$

Randomized

Theorem. *The class **RP** consists of all languages L that have a polynomial-time randomized algorithm A with the following behavior:*
*if $x \in L$, then A accept x in prob at most $\frac{1}{2}$. and if $x \notin L$ A **always** reject*

$$\cup_{c \geq 0} RP(2^{-n^c}) = NP \quad \mathcal{RP}(n^{-c}) = \mathcal{RP}(\frac{1}{2}) = \mathcal{RP}(1 - 2^{-n^c})$$

Theorem. *The class **BPP** consists of all languages L that have a polynomial-time randomized algorithm A with the following behavior:*
if $x \in L$, then A accept x in prob $\geq \frac{3}{4}$. and if $x \notin L$ A accept in prob $\leq \frac{1}{4}$

$$co - \mathcal{RP} = BPP(\frac{1}{2}, 1) \quad BPP(\alpha - n^{-c}, \alpha + n^{-c}) = BPP(2^{-n^c}, 1 - 2^{-n^c})$$

Theorem. *(random algo) Given n digits strings $a, b \in R^n$ and $x \in \{0,1\}^n$ then* $\Pr(ax = bx) \leq 1/2$

Proof. supose $a_j \neq b_j$ Let $\alpha = \sum a_i x_i, \beta = \sum b_i x_i$ when $i \neq j$. then $ax = \alpha + a_i x_i$, $bx = \beta + b_i x_i$ then

$$ax - bx = (\alpha - \beta) + (a_i - b_i)x_i \Pr(ax - bx = 0) = \Pr\left(x_i \frac{\alpha - \beta}{b_i - a_i}\right) \leq 1/2$$

□

Theorem. *(Adelman's Theorem) Every $L \in BPP$ can be decided by a family $\{C_n\}_{n \in N}$ of poly-size circuts*

PP=RP consequences

$$\text{PP} = \text{RP}, \text{coPP} = \text{coRP}, \text{PP} = \text{coPP} = \text{coRP} = \text{RP} = \text{ZPP} = \text{BPP} \subseteq \text{P/poly}$$

If **PP** = **RP** then you have a collapse to the first level, namely **PH** = **NP**. Assume **PP** = **RP**, since **PP** is closed under complement we have **RP** = *co***RP** = **ZPP**, thus **PP** = **ZPP**.
Let L be a language in $\mathbf{PP}^{\mathbf{ZPP}}$, then there exists a probabilistic polynomial time Turing machine with access to a **ZPP** oracle $\mathcal{O}$, call it $M_{\mathcal{O}}$, such that:
$x \in L \Rightarrow \Pr[M_{\mathcal{O}} \text{ accepts x}] > \frac{1}{2}$, and
$x \notin L \Rightarrow \Pr[M_{\mathcal{O}} \text{ accepts x}] \leq \frac{1}{2} - \frac{1}{2^{n^c}}$.
Where n^c is the running time of $M_{\mathcal{O}}$ (in the standard definition, the second inequality appears with $\leq \frac{1}{2}$, however it can be improved to $< \frac{1}{2}$ and thus to the above).
Suppose that the Turing machine which decides $\mathcal{O}$ has expected runtime n^d. Now, let us look at the machine M, which replaces each oracle call of $M_{\mathcal{O}}$ by a t steps simulation of the **ZPP** machine for $\mathcal{O}$, and repeats this simulation k times. If we come upon an oracle call, for which this process did not result in an answer (none of the k t-steps simulations halted), then M rejects.
Let H denote the event that all of the simulations halted in the given time, then:
$\Pr[\text{M accepts x}] = \Pr[\text{M accepts x}|H] \Pr[H] + \Pr\left[\text{M accepts x}\middle|\overline{H}\right] \Pr[\overline{H}] =$
$\Pr[\text{M accepts x}|H] \Pr[H] = \Pr[M_{\mathcal{O}} \text{ accepts x}] \Pr[H]$.
Thus, we have:
$x \in L \Rightarrow \Pr[\text{M accepts x}] > \frac{1}{2} \Pr[H]$, and
$x \notin L \Rightarrow \Pr[\text{M accepts x}] \leq \left(\frac{1}{2} - \frac{1}{2^{n^c}}\right) \Pr[H] \leq \frac{1}{2} - \frac{1}{2^{n^c}}$.

By Markov's inequality, a single simulation does not halt with probability $\leq \frac{n^d}{t}$, so k t-steps simulations do not output an answer with probability $\leq \left(\frac{n^d}{t}\right)^k$. By the union bound we have $\Pr[H] \geq 1 - n^c \left(\frac{n^d}{t}\right)^k \geq 1 - \frac{1}{2^{n^c}}$, for $t = 2n^d$ and $k = 2n^c$. In this case M acheives the following separation:
$x \in L \Rightarrow \Pr[\text{M accepts x}] > \frac{1}{2}\left(1 - 2^{-n^c}\right)$, and
$x \notin L \Rightarrow \Pr[\text{M accepts x}] \leq \frac{1}{2} - 2^{-n^c} < \frac{1}{2} - \frac{1}{2}2^{-n^c}$
Which proves $L \in \mathbf{PP}$, since we can replace the constant $\frac{1}{2}$ with any function $f(x) \in \mathbf{FP}$.

change probability in definition of PP

Lets say a separation using $p \in \mathbb{Q}$ exists for $L \subseteq \Sigma^*$ if there exists a polynomial probabilistic Turing machine M such that:
$x \in L \Rightarrow \Pr[\text{M accepts x}] > p \qquad x \notin L \Rightarrow \Pr[\text{M accepts x}] < p$
$L \in PP$ iff there exists a separation for L using $\frac{1}{2}$ (we can use a strong inequality in both cases).
We proceed to show that $\forall p, q \in (0,1) \cap \mathbb{Q}$, if there exists a separation for L using p, then there exists a separation using q. Now, given a machine M_p with separation p for L, lets consider a machine M_q, which starts by tossing an α-biased coin. If the outcome is 1 then it accepts, otherwise it simulates M_p on the input and returns it's answer.
Since we want M_q to be polynomial in the worst case, we restrict the simulation of the α-biased coin to a certain number of iterations, such that the simulation halts with probability $1 - \epsilon$ (this can be done for any $\epsilon > 0$, simply use the fact that the simulation runs in expected constant time, and apply Markov's inequality). If the simulation didn't halt, M_q accepts. In this case we have:
$\Pr[M_q \text{ accepts x}] = \epsilon + (1 - \epsilon)(\alpha + (1 - \alpha) \Pr[M_p \text{ accepts x}])$, so:
$x \in L \Rightarrow \Pr[M_q \text{ accepts x}] > \epsilon + (1 - \epsilon)(\alpha + (1 - \alpha)p)$
$x \notin L \Rightarrow \Pr[M_q \text{ accepts x}] < \epsilon + (1 - \epsilon)(\alpha + (1 - \alpha)p)$ This means it is enough to require
$q = \epsilon + (1 - \epsilon)(\alpha + (1 - \alpha)p)$. Equivalently, $\alpha = \frac{q - p - \epsilon(1-p)}{1 - \epsilon}$.
Since $p, q \in (0,1) \cap \mathbb{Q}$, if $q > p$ we can find a small rational ϵ and rational $\alpha \in (0,1)$ to satisfy this. If $p > q$ you can change M_q to reject when the outcome of the α-biased coin is 1. When this equality holds, M_q achieves q separation for L and we are done.

CircuitSat $\in$ DTIME(2^n) consequences

CircuitSat $\in$ DTIME(2^n) does not imply NP $\subseteq$ DTIME(2^n) since the reduction might increase the input's size. Suppose for example that $L \in NP$ and the reduction from L to CircuitSat maps strings of length n to strings of length n^c (for all $n \in \mathbb{N}$), then applying the reduction and using a naive exponential time algorithm for circuit sat yields a 2^{n^c} time algorithm for L.
If however you can place an NP-complete problem in a subexponential time class, e.g. $n^{\log n}$, then $NP \neq EXP$, since $n^{O(\log n)}$ is closed under polynomial transformations, i.e. $n \mapsto n^c$.

P/poly = $\cup_{c \in \mathbb{N}}$SIZE(n^c) consequences

If NEXPTIME $\subset$ *P/poly* then NEXPTIME = EXPTIME. **Proof that** $BPP \subseteq P/Poly$ Let L be a language in BPP, and let $M(x, r)$ be a polynomial-time algorithm that decides L with error$\leq 1/3$(where x is the input string and r is a set of random bits).
Construct a new machine $M'(x, r)$, which runs$M, 48n$ times and takes a majority vote of the results (where n is the input length and R is a sequence of $48n$ independently random $r's$). Thus,M' is also polynomial-time, and has an error probability $\leq 1/e$by the (chernof). If we can fix R then we obtain an algorithm that is deterministic.
If Bad(x) is defined as$\{R : M'(x, R) \text{ is incorrect}\}$, we have:
:$\forall x \, Prob_R[R \in \text{Bad}(x)] \leq \frac{1}{e^n}$·¿
The input size is "n", so there are 2¡sup¿"n"¡/sup¿ possible inputs. Thus, by the [[union bound]], the probability that a random "R" is bad for at least one input "x" is
:$Prob_R[\exists x \, R \in \text{Bad}(x)] \leq \frac{2^n}{e^n} < 1$.
In words, the probability that "R" is bad for some "x" is less than 1, therefore there must be an "R" that is good for all "x". Take such an "R" to be the advice string in our "'P/poly"' algorithm.

Prove that PP is closed under complement

algorithm A with the property that $x \in L \Rightarrow \Pr[A(x) = 1] > \frac{1}{2}$ and $x \notin L \Rightarrow \Pr[A(x) = 1] \leq \frac{1}{2}$.
¿ Let $f(|x|)$ be the ¿ polynomial upper bound on the running time of A on input x. Thus A ¿ makes at most $f(|x|)$ random coin flips during its execution. In ¿ particular,
$x \in L \Rightarrow \Pr[A(x) = 1] \geq \frac{1}{2} + \frac{1}{2^{f(|x|)}}$.
I understand that:
- By definition of PP, $x \in L \Rightarrow \Pr[A(x) = 1] > \frac{1}{2}$. It means that if indeed $x \in L$, more than half of the possible coin flip combinations would cause the algorithm to accept. - The algorithm's running time is bound by $f(|x|)$, then the probability of getting some specific combination of coin tosses (that accepts or rejects, we do not know) is bound by $(\frac{1}{2})^{f(|x|)} = \frac{1}{2^{f(|x|)}}$.
But why is it so obvious that $x \in L \Rightarrow \Pr[A(x) = 1] \geq \frac{1}{2} + \frac{1}{2^{f(|x|)}}$

$$x \notin L \Rightarrow \Pr[A' \text{ acc } x] \leq \frac{1}{2} \cdot \left(1 - \frac{1}{2^{f(|x|)+1}}\right) < \frac{1}{2}$$

$$\text{and} x \in L \Rightarrow \Pr[A' \text{ acc } x] \geq \left(\frac{1}{2} + \frac{1}{2^{f(|x|)}}\right) \cdot \left(1 - \frac{1}{2^{f(|x|)+1}}\right) > \frac{1}{2}.$$

This justifies the assumption (since A is still a polynomial-time probabilistic algorithm) and completes the proof.

Theorem. *(extended rice) Let $C \subset RE$ s.t $\Sigma^* \notin C, C \neq \emptyset$. then $L_C \notin RE$*

$L_C = \{\langle M \rangle : M \text{ is TM and} L(M) \in C\} \notin RE$

Proof. $\overline{ACC} \leq_m L_C$ and Let M_A be TM that accept some $A \in C$. Define $f(\langle M, w \rangle) \mapsto \langle B_{M,w} \rangle$,
$B_{M,w}$ On input x: Emulate $M_A(x)$ and $M(w)$ in **parallel**, and **accept** if one of them does.

$$\langle B_{M,w} \rangle \in \overline{ACC} \Rightarrow L(B_{M,w}) = A \in C \qquad \langle B_{M,w} \rangle \notin \overline{ACC} \Rightarrow L(B_{M,w}) = \Sigma^* \notin C$$

□

Rice extended(not the version we saw)

theorem Given a property $S \subseteq RE$, the language

$$L_S = \{\langle M \rangle \mid L(M) \in S\}$$

is recursively-enumerable ($L_S \in RE$) **if and only if** all the following three statements jointly hold
1. For any two $L_1, L_2 \in RE$, if $L_1 \in S$ and also $L_1 \subseteq L_2$ then also $L_2 \in S$.
2. If $L_1 \in S$ then there exists a finite subset $L_2 \subseteq L_1$ so that $L_2 \in S$.
3. The set of all *finite* languages in S is enumerable (in other words: there is a TM that enumerates all the finite languages $L \in S$).

Proof. If (1,2) hold, but (3) does not, then $L_S \notin RE$*. Let's assume that $L_S \in RE$, and we'll see that we have a way to accept any finite languages in S (and thus, the set of all these languages is RE), thus condition (3) holds and we reach a contradiction. How to decide if a finite L belongs to S or not? Easily – we use the description of L to construct a machine M_L that accepts only the words in L, and now we run the machine of L_S on M_L (remember - we assumed $L_S \in RE$, so there is a machine that accepts L_S!). If $L \in S$ then $\langle M_L \rangle \in L_S$ and since $L_S \in RE$, its machine will say yes on the input $\langle M_L \rangle$, and we are done.
If (2,3) hold, but (1) does not, then $L_S \notin RE$*. We assume that $L_S \in RE$ and we'll show that we have a way to decide HP, leading to a contradiction.
Because condition (1) doesn't hold, there is a language $L_1 \in S$ and a superset of it, $L_2 \supseteq L_1$ so that $L_2 \notin S$. Now we are going to repeat the argument used in Section 4 to decide HP: given an input $(\langle M \rangle, x)$ for HP, we construct a machine M' whose language is L_1 if $(\langle M \rangle, x) \notin HP$ or otherwise, its language is L_2. Then, we can decide HP: either M halts on x, or the RE-machine for L_S accepts M'; we can run both in parallel and are guaranteed that at least one will halt.
Let's give the details of constructing M' (on input x'):
1. Do the following in parallel: 1.1 run M on x. 1.2 run the machine of L_1 on x' 2. If 1.2 halts and accepts - accept. 3. If 1.1 halts: run the machine of L_2 on x'.
Why does this work? If $(\langle M \rangle, x) \notin HP$ then 1.1 never halts, and M' accepts exactly all the inputs that are being accepted at step 1.2, so $L(M') = L_1$. On the other hand, if $(\langle M \rangle, x) \in HP$ then, at some point step 1.1 halts and M' accepts exactly L_2. It may happen that 1.2 accepts beforehand, but since $L_1 \subseteq L_2$, this doesn't change the language of M' in this case.
***If (1,3) hold, but (2) does not**, then $L_S \notin RE$*. Again, we will assume $L_S \in RE$ and show that HP becomes decidable, which is a contradiction.
If condition (2) doesn't hold, then for any $L_1 \in S$, all its finite subsets $L_2 \subseteq L_1$ satisfy $L_2 \notin S$ (note that L_1 must be infinite, since $L_1 \subseteq L_1$). As in the above, in order to decide HP for a given input $(\langle M \rangle, x)$, we construct a machine M' whose language is L_1 if $(\langle M \rangle, x) \notin HP$ and some finite L_2 otherwise. The contradiction follows in a similar way as above.
The construction of this machine is quite similar to the previous M' we constructed. The machine M' (on input x') does:
1. Runs M on x for $|x'|$ steps. 2. If M halts during step 1 – reject 3. Otherwise, run the machine of L_1 on x'.
It holds that, if $(\langle M \rangle, x) \in HP$, then at some point, say after 1000 steps, M halts on x. Therefore, step 1 will halt on (and reject) any input x' of length > 1000. Therefore, in this case, $L(M')$ is **finite**. Also note that $L(M') \subseteq L_1$, and in particular, by our assumptions on the invalidity of condition (2), we have that $L(M) \notin S$.
On the other hand, if $(\langle M \rangle, x) \notin HP$, then step 1 never halts, and we never reject at step 2. In this case it is easy to see that $L(M) = L_1$ and in particular, $L(M) \in S$.
We are left to show the other direction of the extended theorem. That is, we need to show that **if all the conditions (1,2,3) hold,** then we have a TM that accepts L_S, that is, $L_S \in RE$. In other words, we need to show a machine M_S so that for any input $\langle M \rangle$ for which $L(M) \in S$, the machine accepts this input, $M_S(\langle M \rangle) \rightarrow$ accept.

Here is how the machine M_S behaves (on input $\langle M\rangle$):
1. Let $M_{\text{enum } S}$ be the machine that enumerates all the finite languages in S, guaranteed by condition (3). 2. Run the following in parallel for $i = 1, 2, ...$ 2.1 Run $M_{\text{enum } S}$ until it outputs the language L_i 2.2. Check if M accepts all the words of L_i (run M on these words, again in parallel). 2.3. If for some i, M accepts all the words of L_i – accept.
Why does it work? If $L(M) \in S$ then it has a finite subset $L_j \in S$, and once $M_{\text{enum } S}$ outputs that subset, step 2.2/2.3 will find that M accepts all the words in that language and accept.
On the other hand, if $L(M) \notin S$ it cannot be accepting all the words in L_i for any $i = 1, 2, ...$
Indeed, by condition (1), any $L' \supseteq L_i$ is also in S, so if M accepts all the words in L_i for some i, then $L(M) \supseteq L_i$ and thus $L(M) \in S$, in contradiction.

Given a non trivial property $S \subsetneq RE$, so that $\emptyset \in S$, the language ¿

$$L_S = \{\langle M\rangle \mid L(M) \in S\}$$

¿ is not recursively-enumerable, that is, $L_S \notin RE$.

Stuff that legit to use from HW

claim. *For language L s.t L satisfies pumping lemma with pumping constant ℓ , $L||L$ satisfies pumping lemma with pumping constant 2ℓ*

$L' = \{xyz : xy^R z \in L\}$ is regular if L regular
$L' = \{xy \in \Sigma^* : (x \in L) \text{ XOR } (y \in L)\}$ is regular regular if L regular.

Proposition 1. *$\mathcal{RE}$ is **not** closed under complement*

Proposition 2. *co$\mathcal{RE}$ is closed under intersection.*

Proposition 3. *$\mathcal{R}$ is closed under Kleene star.*

Proposition 4. *$\mathcal{RE}$ is closed under Prefix, but $\mathcal{R}$ is **not** closed under Prefix,*

(A) **Prove:** If $\mathcal{L}_1 \leq_m \mathcal{L}_2$ and $\mathcal{L}_2 \leq_m \mathcal{L}_3$, then $\mathcal{L}_1 \leq_m \mathcal{L}_3$.
(B) **Disprove:** If $\mathcal{L}_1 \leq_m \mathcal{L}_2$ and $\mathcal{L}_2 \leq_m \mathcal{L}_1$, then $\mathcal{L}_1 = \mathcal{L}_2$.
(C) **Disprove:** If $\mathcal{L}_1 \subseteq \mathcal{L}_2$ then $\mathcal{L}_1 \leq_m \mathcal{L}_2$.
(D) **Disprove:** For every $\mathcal{L}_1, \mathcal{L}_2$, then $\mathcal{L}_1 \leq_m \mathcal{L}_2$ or
(E) **Prove:** If $\mathcal{L}$ is regular, then $\mathcal{L} \leq_m HALT$

(a) Input: Sets $A_1, ..., A_n$, and a number k.
Question: do there exist k mutually disjoint sets $A_{i1}, ..., A_{ik}$ **not in $\mathcal{P}$**

(b) Input: Sets $A_1, ..., A_n$.
Question: do there exist 3 mutually disjoint sets A_i, A_j, A_k **in $\mathcal{P}$**

(c) Input: Sets $A_1, ..., A_n$, and a number k.
Question: do there exist k sets such that $A_{i1}, ..., A_{ik}$ such that $A_i \cap A_j \neq \emptyset$ **not in $\mathcal{P}$**

(d) Input: a CNF formula ψ with at most 50 variables.
Question: does there exist an assignment that satisfies ψ **in $\mathcal{P}$**

(e) Input: a CNF formula ψ with at most 50 clauses.
Question: does there exist an assignment that satisfies ψ **in $\mathcal{P}$**

(f) Input: a 3CNF formula ψ with even number of variables.
Question: does there exist an assignment that satisfies ψ and gives $True$ for exactly one half of the variables? **not in $\mathcal{P}$**

(g) Input: undirected graph G, a number k.
Question: does there exist in G a clique of size at least k or an independent set of size at least k? **not in $\mathcal{P}$**

(a) For every nontrivial $L_1, L_2 \in P, L_1 \leq L_2$ **TRUE**

(b) For every nontrivial $L_1, L_2 \in NP, L_1 \leq_P L_2$ **Equivalent to an Open Problem**

(c) There exists a language in $\mathcal{RE}$ that is complete w.r.t polynomial-time reductions. **TRUE.**

(d) If there exists a deterministic TM that decides SAT in time $n^{O(\log n)}$
Then every $L \in NP$ is decidable by a deterministic TM in time $n^{O(\log n)}$. **TRUE.**

$EXACT3SAT = \{\varphi \in 3SAT$: every clause of φ has exactly 3 distinct variables$\}$ is **NPC**
$L_2 = \{\langle M, 1^n\rangle$: M is a TM and there exists a string that M accepts in n steps$\}$is **NPC**

Resulting regular expression:

$L' = \{0^i 1^j : i, j \geq 0$ and $\exists w \in Ls.t|w| = i + j\}$ is regular if L regular **T**. built NFA
$Q \times \{0, 1\}, \delta'((q, 0), 0) = \{(\delta(q, \sigma), 0) : \sigma \in \Sigma^*\}, \delta'((q, 0), \epsilon) = \{(\delta(q, 1), \sigma) : \sigma \in \Sigma^*\}$
$L' = \{0^i 1^j : i, j \geq 0$ and $\exists w \in Ls.t|w| = |i - j|\}$is regular if L regular **F**. take $L = \epsilon$
r = r1* r2(r4+r3r1*r2)* where: r1: initial → initial r2: initial → accepting r3: accepting → initial r4: accepting → accepting

$$L_2 = \{y : \exists x, z \;\; s.t.|x| = |z| \; and \; xyz \in L\}$$

(Example) is regular.onsider some DFA for L. For every $n \geq 0$, let A_n be the set of states s such that there is *some* word of length n which leads the DFA from the initial state to s. Let B_n be the set of states t such that there is *some* word of length n which leads the DFA from t to an accepting state. Finally, for any two states s, t, let $R_{s,t}$ be the (regular) set of words leading the DFA from s to t. We have

$$L_2 = \bigcup_{n \geq 0} \bigcup_{\substack{s \in A_n \\ t \in B_n}} R_{s,t}.$$

Since there are only finitely many possibilities for s, t, the union is in fact finite, and so regular.

Hope to see one of them in the exam

$\text{NTIME}(n^k) \subsetneq \text{NTIME}(n^k \log(\log n)) \subsetneq \text{NTIME}(n^k \log n) \subsetneq \text{NTIME}(n^{k+1}) \quad k \geq 1$
$DTIME(n^k) \subsetneq DTIME(n^k \log(\log n)) \subsetneq DTIME(n^{k+1}) \quad k \geq 1$

1. $\mathcal{L} = \{\langle M\rangle : M$ is a TM and $|L(M)| > 10\} \in \mathcal{RE}/\mathcal{R}$
2. $\mathcal{L} = \{M : \langle M\rangle$ is a TM that accepts 1 but does not accept $0\} \in \overline{\mathcal{RE} \cup \text{co-}\mathcal{RE}}$
3. $\mathcal{L} = \{\langle M_1, M_2\rangle : M_1, M_2$ are TMs and $\mathcal{L}(M_1) \cap \mathcal{L}(M_2) = \emptyset\} \in$ co-$\mathcal{RE}/\mathcal{R}$
4. $\mathcal{L} = \{\langle M_1, M_2\rangle : M_1, M_2$ are TMs and $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)\} \in \overline{\mathcal{RE} \cup \text{co-}\mathcal{RE}}$
5. $L_n = \{w0w_n | w \in \{0,1\}^* \wedge w_n \in \{0,1\}^{n-1}\}$ is Regular
6. $L_1 = \{w : \#a(w) \geq \#b(w)\}$ over $\Sigma = \{a, b, c\}$ L_1 is not regular
7. $L_2 = \{w : |w| \in \mathbb{N}_{even} \wedge w = w^R)\}$ over L_2 is not regular
8. $L_4 = \{w : |w| \in \mathbb{N}$ s.t $|w| = n^3\}$ over L_4 is not regular
9. $L = \{\langle M\rangle | M$ is a TM and there exists an input on which M halts in less than $|M|$ steps $\} \in R$
10. $L = \{\langle M\rangle | M$ is a TM and $|L(M)| \leq 3\} \in$**co-RE** reduce $\overline{HALT}$. idea copies M and x to its tape, and runs M on x; it accepts if M halts on x.
11. $L = \{\langle M\rangle | M$ M is a TM and $|L(M)| \geq 3\} \in$ **RE** reduce $HALT$. It erases w, puts M and x on its tape, and runs M on x and accepts if M halts on x.
12. $L = \{\langle M\rangle | M$ is a TM that accepts all even numbers $\} \in$**not RE**
13. $L = \{\langle M\rangle | M$ is a TM and $L(M)$ is finite $\} \in$**not RE**
14. $L = \{\langle M\rangle | M$ is a TM and $L(M)$ is infinite $\} \in$**not RE**
15. $L = \{\langle M\rangle |$ is a TM and $L(M)$ is countable $\} \in$**R** This is the language of all TM's, since there are no uncountable languages (finite alphabets and finite-length strings).
16. $L = \{\langle M\rangle | M$ is a TM and $L(M)$ is uncountable $\} \in$ **R** This is the empty set
17. $L = \{\langle M_1, M_2\rangle |$ are TM's and $\epsilon \in L(M_1) \cup L(M_2)\} \in$ **RE**. reduce HALT s.t $\langle M, x\rangle \mapsto \langle M', M'\rangle$ $\langle M, x\rangle$ halts on $x \Rightarrow M'$ accepts all strings, and in particular it accepts $\epsilon \Rightarrow \epsilon \in L(M') \cap L(M')$.
18. $L = \{\langle M_1, M_2\rangle |$ are TM's and $\epsilon \in L(M_1) \cap L(M_2)\} \in$ **RE**.
19. $L = \{\langle M_1, M_2\rangle |$ are TM's and $\epsilon \in L(M_1)/L(M_2)\} \in$ **not RE**.
$\overline{HALT}(M, x) \mapsto (M_1, M_2)$. s.t $L(M_1) = \Sigma^*$
20. $L = \{\langle M\rangle | M$ is a TM. and M_0 halt on all input, and $M_0 \in L(M)\} \in$ **RE** using Rice $C = \{L \in RE : M_0 \in L\}$
21. $L = \{\langle M\rangle | M$ is a TM. and M_0 halt on all input, and $M \in L(M_0)\} \in$ **R**
22. $L = \{\langle M\rangle | M$ is a TM. and x is str and there exists a TM, M' , such that $M \notin L(M') \cap L(M)\} \in$ **R**
23. $L = \{\langle M\rangle | M$ is a TM. , and there exists an input whose length is less than 100, on which M halts $\} \in$. **RE**
24. $L = \{\langle k, x, M_1, \dots M_k\rangle | \forall i M$ is a TM, and at least $k/2$ TM's of halt on x $\} \in$. **RE** its $\emptyset$ reduce $HALT(M, X) \mapsto (2, x, M, M')$
25. $L = \{\langle M\rangle | M$ is a TM and M halts on all palindromes .$\} \in$.**not RE** .$\overline{HALT}(M, w) \mapsto M'$ on input x works as follows: M' runs M on w for $|x|$ steps: if M halts on w within $|x|$ steps, reject, otherwise accept.(if x is polindrome)
26. $L = \{\langle M\rangle | M$ is a TM and $L(M) \cap \{a^{2^n}\}\}$ is empty.$\} \in$.**not RE** $\overline{HALT}(M, w) \mapsto M'$ M' on input x: erase x, and run M on w. If M halts on w then M' accepts.
27. $L = \{\langle M\rangle | M$ is a TM that halts on all inputs and $L(M) = L'$ for some undecidable language L' .$\} \in$.**R**
28. $L = \{\langle M\rangle | M$ is a TM and M accepts (at least) two strings of different lengths .$\} \in$.**RE**
29. $L = \{\langle M\rangle | M$ M is a TM such that both $L(M)$ and $\overline{L(M)}$ are infinite .$\} \in$.**not RE** M' on input x works as follows: if x is of odd length, accept. if x is of even length, run M on w. If M halts, accept.
$(M, w) \in \overline{HALT} \Rightarrow M$ does not halt on $w \Rightarrow M'$ accept all odd length str and reject all even.$\Rightarrow L(M)$ contain even-length strings,$|L| = \inf$
30. $L = \{\langle M\rangle | M$ is a TM, and $|L(M)|$ is prime.$\} \in$.**not RE** M' on input x: if x is one of the first three strings (in lexicographic order) of $\Sigma*$, then accept. Otherwise, run M on w. If M halts on w, and x is the 4 th string in $\Sigma*$ accept; otherwise, reject.
31. $L = \{\langle M\rangle | M$ and exists x s.t for every $y \in L(M), xy \notin L(M).\} \in$.**not RE** M' on input x: it runs M on w; if M halts on w, M' accepts.
32. $L = \{\langle M\rangle | M$ and exists x, y s.t for every $y \in L(M), y \notin L(M).\} \in$.**Recursive** (the language of all Turing machines)
33. $L = \{\langle M\rangle | M$ is TM and exists M' s.t $M \neq M'$ and $L(M) = L(M').\} \in$.**Recursive**(the language of all Turing machines)
34. $L = \{\langle M_1, M_2\rangle | M$ and $L(M_1) \leq_m L(M_2)\}$**not RE** $\overline{HT}(M, w) = \langle M_1, M_2\rangle$. s.t $L(M_2) = \emptyset$. M_1 on input y: run M on w if M halts, check whether y is of the form $\langle M', z\rangle$ where M' is TM and z is str. if so run M' on z. if M' halts, M_1 accept . hence (1)if $(M, w) \in \overline{HT} \Rightarrow L(M_1) = \emptyset$ (2) otherwise then $L(M_1) = HALT$
35. $L = \{\langle M_1, M_2\rangle | M$ M does not accept any string w such that 001 is a prefix of $w\}$**not RE**
36. $L = \{\langle M\rangle | M$ M does not accept any string w s x is a prefix of $w\}$**not RE**
37. $L = \{\langle M\rangle | M$ x is prefix of $M\}$**Recurcive**
38. $L = \{\langle M_1, M_2, M_3\rangle | L(M_1) = L(M_2) \cup L(m_3)\}$**not RE**
39. $L = \{\langle M_1, M_2, M_3\rangle | L(M_1) \subseteq L(M_2) \cup L(m_3)\}$**not RE** M1 on x: runs M on w; if M halts, M1 accepts.M2 on y: reject. M3 on z: reject.
40. $L = \{\langle M_1\rangle$ there exist two TM's M2 and M3 such that $|L(M_1) \subseteq L(M_2) \cup L(m_3)\}$**Recursive**
41. $L = \{\langle M, w\rangle$ M is a TM that accepts w using at most $2^{|w|}$ squares of its tape$\}$**Recursive** Let m be the number of states in M, and k be the size of the alphabet that M uses, and $r = |w|$ if M uses at most 2^r squares of its tape, then there are at most $a = mk^{2^r} 2^r$ configurations.If M runs on w for more than a steps, and does not use more than 2^r squares of its tape, then M must be in the one configuration at least twice.
M' runs M on w for at most a + 1 steps.If M accepts w within a steps with using at most 2^r squares, m' halts and accepts

Random stuff

1) Wich one is **False** for sure $\underline{DTIME[n^2] = L} \quad P = NP \quad RL \in P$
2)Wich one we dont know if it close under klee star $\underline{L} \quad P \quad NP \quad EXP$
3) Let f be computable with circut size $|S|$ we can construct circuit size $O(S)$ with only AND,NOT gate for sure.
4) $L = \{G|$G is an undirected graph that contains no odd length cycle$\} \in RL, NL$(mybe)
5) $L = \{G, 1^k|$ G is an undirected with exactly k triangles$\} \in L$
6) $L = \{\psi$ is (always true) $\} \in Co - NP$ complete
7) $L = \{G, k|$ G is an undirected with exactly k diffrent SCC$\} \in SPACE(\log^2 n), P$
8) If we find that $S = \{G, s, t$ dont have path between s and t $\} \in L$ then $L = Co - NL$
$\{\langle G, s, t\rangle$ G is an directed graph, and the shortest patch between s, t is at **least** size $|V|/2\} \in NL$
$\{\langle G, s, t\rangle$ G is an directed graph, and exsit patch between s, t **excatliy** size $|V|/2\} \in NPC$
9) If $\text{NTIME}(n^{100}) \subseteq \text{TIME}(n^{1000})$ then $P = NP, EXP = NEXP, \text{NSPACE}(n^{50}) \subseteq \text{SPACE}(n^{500})$

Bali Bnangim

♠ stand for respect under logspace reduction
CVAL=$\{\langle C, x\rangle$: C is boolean circut and $C(x) = 1\} \in \mathcal{P}-$complete ♠
STCON=$\{\langle G, s, t\rangle$: G is a directed graph and exsit path from s to t $\} \in \mathcal{P}, NL-$complete ♠
UHAMPATH=$\{\langle G, s, t\rangle$: G is a undirected graph and exsit ham' path from s to t $\} \in NPC$
UHAMCYCLE=$\{\langle G\rangle$: G is a undirected graph has Hamiltonian cycle $\} \in NPC$
SUBSET-SUM=$\{\langle S = s_1 \dots s_k, t\rangle : \exists T \subseteq S, \sum_{s \in T} s = t\} \in NPC$

Theorem. *(extended rice) Let $C \subset RE$ s.t $\Sigma^* \notin C, C \neq \emptyset$. then $L_C \notin RE$*

$L_C = \{\langle M\rangle : M$ is TM and$L(M) \in C\} \notin RE$

Proof. $\overline{ACC} \leq_m L_C$ and Let M_A be TM that accept some $A \in C$. Define $f(\langle M, w\rangle) \mapsto \langle B_{M,w}\rangle$, $B_{M,w}$ On input x: Emulate $M_A(x)$ and $M(w)$ in **parallel**, and **accept** if one of them does.

$$\langle M, w\rangle \in \overline{ACC} \Rightarrow L(B_{M,w}) = A \in C \qquad \langle M, w\rangle \notin \overline{ACC} \Rightarrow L(B_{M,w}) = \Sigma^* \notin C$$

□

2-NFA with same L

to compare languages accepted by both we have to figure out if L(A) is equal to L(B) or not. Thus, you have to find out if L(A)-L(B) and L(B)-L(A) is null set or not. (Reason1)
Part1: To find this out, we construct NFA X from NFA A and NFA B, . If X is empty set then L(A) = L(B) else L(A) != L(B). (Reason2)
Part2: Now, we have to find out an efficient way of proving or disproving X is empty set. When will be X empty as DFA or NFA? Answer: X will be empty when there is no path leading from starting state to any of the final state of X. We can use BFS or DFS for this.$X = (A \cap B') \cup (B \cap A') = \emptyset$?

(a) $L = \{\langle M, w\rangle M$attempts to move its head left when its head is on the leftmost tape cell$\}$

We prove by contradiction that L is undecidable. Assume L was decidable, and let $M*$ be a TM that decides it. We construct a TM, M0 , that decides HP (the halting problem), i.e., we show HP is decidable as well, which is a contradiction. The TM M0 on input hM, wi: i. build a TM, M00 , from M, where M00 shifts w one tape cell to the right, and mark the leftmost tape cell with]. ii. TM M00 runs M on w. iii. If M00 encounters the] symbol during the run of M on w, then M00 moves to the right and simulates M reaching the leftmost tape cell. iv. If M halts and accepts, M00 simulates moving its head all the way to the left and off the leftmost tape cell The TM M0 runs $M*$ on the input hM0 , wi. If $M*$ accepts, M0 accepts; otherwise, M0 rejects. Notice that $M*$ always halts (accepts or rejects) since we assumed it is a decider. You can prove now that M00 moves its head off of the leftmost tape cell iff M halts (and accepts) w. It then follows that M0 decides HP, and hence HP is decidable, which is a contradiction. Therfore, the language L is not decidable.

(b) $L = \{\langle M, w\rangle M$moves its head left on input $w\}$

The trick to proving this language is decidable is to notice that M moves its head left on an input w if and only if it does so within the first $|w| + |Q| + 1$ steps of its run on w, where $|Q|$ is the number of states of the machine M. Prove it! Therefore, to decide whether an input hM, wi is an instance of L, the decider$M*$ simply runs M on w for at most $|w| + |Q| + 1$steps and accepts iff M does moves its head left (the correctness of this construction follows immediately after you prove the validity of the above trick)

$ACCBS = \{\langle M, w, k\rangle M$ is a TM that accepts w using k cells$\} \in R$

$b = |Q| \times |\Gamma|^k \times k$ is a bound on the number of k cell configurations of M.

Any $f(n)$ space bounded Turing machine also runs in time $2^{O(f(n))}$. Is this because a configuration consists of a state, a position of the head and the contents of the work tape, which is $|Q| \cdot f(n) \cdot |\Gamma|^{f(n)}$ If there are k possible configurations, then any such Turing machine either runs in time at most k, or it loops forever. That's because if it runs for at least $k+1$ time steps, some configuration must be repeated; and if a configuration is repeated, it will continue repeating forever. Let $|\Gamma| = 2^c$, i.e., $c = \lg |\Gamma|$. Then $|\Gamma|^{f(n)} = (2^c)^{f(n)} = 2^{cf(n)} = 2^{O(f(n))}$. For similar reasons, $|Q| \cdot f(n) \cdot |\Gamma|^{f(n)} = 2^{O(f(n))}$. So, the number of possible configurations of a $f(n)$-space-bounded Turing machine is $2^{O(f(n))}$, and thus any such machine must either halt in time at most $2^{O(f(n))}$ or loop forever.

$\{\langle M \rangle \mid M$ is a Turing machine such that $L(M) \in P\}$ is undecidable.

$\{\langle M \rangle \mid M$ is a Turing machine such that $M \in P\} = \{\langle M \rangle \mid (\exists k \forall x) M(x)$ halts in $O(|x|^k)$ time$\}$

We reduce from the halting problem. Suppose that we are given a machine M, and we are to decide whether M halts on the empty input. We construct a new machine M' accepting a single input x, which operates as follows:
1. Let $n = |x|$. 2. M' runs M for n steps. 3. If M halted within n steps then M' runs a dummy loop taking exponential time $\Omega(2^n)$. Otherwise, M' just halts.
Since Turing machines can be simulated with only polynomial overhead, if M doesn't halt then M' runs in polynomial time. If M halts, then M' takes exponential time. Hence M halts iff M' is not polynomial time.

—

More generally, this shows that even if we know that M runs in time at most $f(n)$ for some superpolynomial time-constructible f, then we cannot decide whether M runs in polynomial time.

Proof.

$$\phi = \bigwedge_{m=1}^{n} (x_m \vee y_m \vee z_m) \qquad \mathsf{SAT} \leq_p \mathsf{IndSet}$$

with m clauses, produce the graph G_ϕ that contains a triangle for each clause, with vertices of the triangle labeled by the literals of the clause. Add an edge between any two complementary literals from different triangles. Finally, set $k = m$. In our example, we have triangles on $x, y, \overline{z}$ and on $\overline{x}, w, z$ plus the edges $(x, \overline{x})$ and $(\overline{z}, z)$.
¿We need to prove two directions. First, if ϕ is satisfiable, then G_ϕ has an independent set of size at least k. Secondly, if G_ϕ has an independent set of size at least k, then ϕ is satisfiable. (Note that the latter is the contrapositive of the implication "if ϕ is not satisfiable, then G_ϕ does not have an independent set of size at least k".)
¿For the first direction, consider a satisfying assignment for ϕ. Take one true literal from every clause, and put the corresponding graph vertex into a set S. Observe that S is an independent set of size k (where k is the number of clauses in ϕ).
¿For the other direction, take an independent set S of size k in G_ϕ. Observe that S contains exactly one vertex from each triangle (clause) , and that S does not contain any conflicting pair of literals (such as x and $\overline{x}$, since any such pair of conflicting literals are connected by an edge in G_ϕ). Hence, we can assign the value True to all the literals corresponding with the vertices in the set S, and thereby satisfy the formula ϕ.
¿This reduction is polynomial in time because ...?
I've looked at many different examples of how this is done, and everything I find online includes everything in the proof except the argument of why this is polynomial. I presume it's being left out because it's trivial, but that doesn't help me when I'm trying to learn how to explain such things.

□

$L = \{\langle M, 1^n \rangle$: M is a TM and exists a string that M accepts in n steps$\}$ (cook-levin)

a.$L \in NPC$ The witness is a word that M accepts in at most n steps b. $L' \in NP$ and let V be a verifier for L' of running time at most p for some polynomial p. For a string w, let v_w be the TM that on input c, emulates $V(w, c)$. Consider the reduction $f(w) = (\langle V_w, 1^{p(|w|)} \rangle$ It is easy to verify that f is polytime computable, and that $f(w) \in L \leftrightarrow w \in L$

3-SAT: Given a CNF formula φ, where every clause in φ has *exactly* 3 literals in it, one should determine if there exist an assignment that satisfies it.

Max-2-SAT: Given a CNF formula, where every clause in ϕ has *exactly* 2 literals in it, and a positive number k, one should determine if there exist an assignment that satisfies *at least* k clauses.

Proof. Replace a singleton clause a with $a \vee b$ and many copies of $\neg b \vee c$, $\neg b \vee \neg c$. Reductions without extra variables don't work even when singleton clauses are allowed. There are five types of clauses: $\ell_1, \neg\ell_1, \ell_1 \vee \ell_2$, $\ell_1 \vee \neg\ell_2$, $\neg\ell_1 \vee \neg\ell_2$. Suppose that k of ℓ_1, ℓ_2, ℓ_2 are true. The probability that a random clause of each type is true is:

k	0	1	2	3
ℓ_1	0	1/3	2/3	1
$\neg\ell_1$	1	2/3	1/3	0
$\ell_1 \vee \ell_2$	0	2/3	1	1
$\ell_1 \vee \neg\ell_2$	1	1/3	1/3	1
$\neg\ell_1 \vee \neg\ell_2$	1	1	2/3	0

We would like to take a convex combination of these rows which is of the form

$$\alpha \quad \beta \quad \beta \quad \beta$$

However, all such convex combinations have $\alpha = \beta$. □

$3SAT \leq_P$ exactly 1/3 SAT

Proof. Let z be a new variable not in ϕ. Let there be m clauses in ϕ. We add $(z \vee \neg z)$ and $2m+2$ copies of (z) to ϕ. In every exactly 1/3-satisfying assignment z must be false.
Let $\phi \wedge (z \vee \neg z) \wedge (z) \wedge (z) \wedge ...(2m+2)$ copies $... \wedge (z) = \phi'$. The following claim is easy to prove.
Claim: ϕ is satisfiable iff ϕ' has exactly 1/3rd satisfiable clauses.
Therfore 3SAT $\leq_P$ 1/3-CNFSAT, and hence 1/3-CNFSAT is NP-complete.

□