Alin Tomescu, *6.840 Theory of Computation (Fall 2013)*, taught by Prof. Michael Sipser

Time and space classes

Little Oh (o, <) and Big Oh (O, $\leq$)

Quick description:

$$f(x) \in O(g(x)) \Leftrightarrow f(x) \leq cg(x), \text{for some } c \text{ and for all big enough } x$$

$$f(x) \in o(g(x)) \Leftrightarrow \lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$$

Also equivalent to saying that for all $c > 0$, there exist big enough x such that $f(n) < cg(n)$

What we know

Theorem	*Details*
$P \subseteq NP \cap coNP$	$\forall A \in P, A \in NP \wedge \bar{A} \in NP$
NP-complete $\subset$ *NP-hard*	By definition of NP-hard, every NP-complete problem is also NP-hard.
$NL \subseteq P$	
$NL \neq PSPACE \neq EXPSPACE$	Space hierarchy theorem.
$P \neq EXPTIME$	Time hierarchy theorem.
$UNIQUE\text{-}SAT \in P^{SAT}$	If we can find an $x_i, i \leq n$ such that $\phi_i(x_1 = a, x_2 = b, \dots, x_i, x_{i+1}, \dots, x_n)$ is satisfiable for $x_i = 0$ and $x_i = 1$ then we reject. Otherwise, we recurse on the x_i assignment that made ϕ satisfiable.
$IP = PSPACE$	
$NP \subseteq IP$	Prover sends certificate, verifier runs the verification algorithm.
$BPP \subseteq IP$	Prover does nothing, verifier runs the BPP algorithm for the problem in poly-time. Or because $BPP \subseteq PSPACE$ since we can simulate all possible coin flips in NPSPACE and thus simulate a prob. poly time TM.
$weak\text{-}IP = BPP$	Because, just like with IP, we can solve any BPP problem in $weak\text{-}IP$ by just having the verifier run the BPP algorithm. Thus, $BPP \subseteq weak\text{-}IP$. Also, any problem in $weak\text{-}IP$ can be solved in BPP since both the prover and verifier are polynomial, which means their interaction will be polynomial, and can be simulated in BPP.

Complexity classes

- RED means the problem is known to be complete for its class
- PURPLE means the problem is not known to be complete for its class (but could be, open for proving)
- BLACK means I have not investigated ☺

Problems in ***P***	*Reasons or explanation*
PATH	Run a BFS/DFS on the graph.
RELPRIME	Run the Euclidean algorithm: *Until* $y = 0$, $x \leftarrow x \bmod y$, exchange x and y. Output x.
PRIMES	Use the recently discovered AKS algorithm.
A_{CFG}	Convert grammar to CNF. Use dynamic programming $O(n^4)$ algo. to build $n \times n$ table for acceptance.
A_{DFA}	Simulate the DFA and see if w is accepted.

Alin Tomescu, *6.840 Theory of Computation (Fall 2013)*, taught by Prof. Michael Sipser

E_{DFA}	Run a BFS/DFS on the DFA's graph to see if there is a path from start to accept state.
E_{NFA}	Similar to E_{DFA}.
ALL_{DFA}	Build the complement of the DFA, by flipping accept into non-accept states and vice-versa.
EQ_{DFA}	Take the symmetric difference, get an NFA, and use $E_{NFA} \in P$ on it.

Problems in ***NP***	*Reasons or explanation*
HAMPATH	The certificate c is the Hamiltonian path. We can check it against the graph G.
LPATH	Reduction from $HAMPATH$.
COMPOSITES	c is the two numbers a, b such that $n = ab$.
CLIQUE	c is the k-clique itself. We can check it against the graph G.
SUBSET-SUM	$\{\langle S, t\rangle \mid S = \{x_1, \dots, x_k\}$, and for some $\{y_1, \dots, y_k\} \subseteq S$, we have $\sum y_i = t\}$, $c = \{y_1, \dots, y_k\}$.
SAT, 3SAT	c is the truth value assignments of the variables such that $\phi = true$
VERTEX-COVER	Every edge in G needs to touch the nodes in the VC. c is the set of nodes of size k in the VC.
ISO	Graph isomorphism $\langle G_1, G_2\rangle$, because $c =$ permutation of nodes
FACTORING	In $NP \cap coNP$. Not known to be NP-complete. If proven to be (surprisingly) then $NP = coNP$.
$\overline{EQ_{BP}}$	Non-deterministically try all assignments, accept if the two BPs disagree on one. coNP-complete.
$\overline{EQ_{ROBP}}$	Not known to be coNP-complete, but can be solved using reduction to $\overline{\text{EQ}_{\text{BP}}}$.

Remember: A is NP-complete and $A \leq_p \bar{A} \Rightarrow NP \subseteq coNP$

Problems in ***PSPACE***	*Details*
TQBF	Basically an instance of SAT, with "exists" (∃) quantifiers. To solve it in PSPACE, we have to take care of "for all" (∀) quantifiers by trying all truth values for such variables and making sure they all satisfy the expression. We can do this recursively, peeling off each quantifier and filling in truth values in the expression, until we can evaluate it ($O(n + \log n)$ space). PSPACE-completeness proof is similar to Savitch's theorem proof, but we rely on the quantifiers to assert the existence of an intermediary configuration of the PSPACE machine we are trying to simulate.
A_{LBA}	We can solve $TQBF$ using an LBA and a recursive algorithm, filling in the values for each variable. (see HW6 for $TQBF \leq_p A_{LBA}$)
FORMULA-GAME	$\exists x_1, \forall x_2, \exists x_3[(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3)]$, each player takes turns picking a value for the variables. Can the first player win? Prove it's PSPACE-complete using reduction from TQBF (insert dummy variables with appropriate quantifiers to convert any QBF to a formula game).
GG	Given a directed graph G and a start node s, players take turns picking the next node of *a simple path* (respecting graph transitions, no node repeats). The goal of player A is to get player B "stuck" in a node, such that he has no more edges to pick for his next path (all used up, or none outgoing). Prove it's PSPACE-complete using reduction from TQBF or FORMULA-GAME.
EQ_{NFA}	We can try all strings of size $2^{\max\{\lvert Q_1\rvert, \lvert Q_2\rvert\}}$ (non-deterministically by picking next symbol and next transitions in NFAs to simulate) and see if the two NFAs disagree on any string. If they do, then they're not equal.
EQ_{REX}	Convert to NFAs and run EQ_{NFA}.
ALL_{NFA}	$\overline{ALL_{NFA}} \in NPSPACE$ because we can non-deterministically try all strings of length $\leq 2^{\lvert Q\rvert}$ and see if the NFA rejects. From Savitch $\Rightarrow \overline{ALL_{NFA}} \in PSPACE$ and, since $PSPACE = coPSPACE \Rightarrow ALL_{NFA} \in PSPACE$.

MIN-FORMULA	We don't care about time, so we can generate all small formulas (a lot of them), one at a time, and check if the two formulas are equivalent (exponential time). Not known to be in $coNP$.

Problems in ***EXPSPACE***	*Reasons or explanation*
$EQ_{REX\uparrow}$	Regular expressions with exponentiation. Reduction from M, w is $R_1 = \Delta^*, \Delta = Q \cup \Gamma \cup \{\#\}$ and $R_2 =$ all strings over Δ that are not rejecting computation histories of M on w. $R_1 \equiv R_2$ iff. M accepts w.

Problems in ***L***	*Details*
A_{DFA}	Simulate DFA by keeping track of current node in DFA, a number from 1 to n, representable on $\log n$ bits.
UPATH	Complicated proof in a paper quoted in the textbook.

Problems in ***(co)NL***	*Reasons or explanation*
PATH	In NL, because we can non-deterministically pick the next node and guess the path (repeat n times). It's complete because we can reduce any NL problem to PATH in log-space by creating a graph of all the configurations with transitions between possible adjacent configurations. Note that a configuration for an NL machine is log-sized, so our log-space transducer can work.
$\overline{PATH}$	Fairly non-obvious algorithm in NL. See resources online and in textbook. $\Rightarrow NL = coNL$.
E_{DFA}	Show E_{DFA} is coNL-complete using reduction from $\overline{PATH}$. This will imply it's also NL-complete.
E_{NFA}	Also by reduction from $\overline{PATH}$.
A_{NFA}	We can non-deterministically simulate the NFA up to $\lvert w\rvert \times \lvert Q\rvert$ transitions to see if it accepts w. We can reduce from $PATH$, by creating an NFA just like the original graph but with just ε transitions and $q_0 = s, q_{acc} = t$. We can check if the NFA accepts the empty string.
$BOTH_{NFA}$	$\{\langle M_1, M_2\rangle \mid$ NFAs where $L(M_1) \cap L(M_2) \neq \emptyset\}$ is in NL because you can try all strings of size $\leq k_1 k_2$ (# of states in the NFAs) and see if they agree on any. It's complete because we can reduce $\overline{E_{DFA}}$ to it by using M_2 as the NFA for Σ^*.
EQ_{DFA}	Because we can decide $\overline{EQ_{DFA}}$ if we try all strings of length $\leq \max\{k_1, k_2\}$ (number of states) and see if they disagree on anything.
CYCLE	Can solve with PATH from s to s. Can reduce from PATH by "unwinding" the graph: 1) $\forall v \in V$, add $v_k \in V_k$, 2) $\forall a \to b \in E$, add $\forall k, a_{k-1} \to b_k$ in G', 3) add $t_{\lvert V\rvert} \to s_1$ in G'.
ODDCYCLE	We can non-deterministically select a start vertex, an odd length $l \leq \lvert V\rvert$ and then keep guessing non-deterministically for the next node. If there is an odd cycle, we will guess it.
BIPARTITE	Can use the $ODDCYCLE$ algorithm to make sure there are no odd cycles in the graph.

Remember: For $NL/coNL$, when proving something is in NL you can either:

- Prove it's in $coNL$, and $NL = coNL$
- Logspace-reduce it to another problem in $NL/coNL$, like $DISJOINT_{REX} \leq_l DISJOINT_{NFA} \in NL$

Unknown problems	*Reasons or explanation*
$\overline{HAMPATH}$	We cannot come up with a certificate to prove it's in NP.
EQ_{NFA}	Not known to be in P, according to prob. 7.11 in 3rd ed.

Probabilistic complexity classes

A probabilistic time TM M decides language A with error probability ε if for all w, we have:

- $w \in A \Rightarrow \Pr[M \text{ accepts } w] \geq 1 - \varepsilon$
- $w \notin A \Rightarrow \Pr[M \text{ accepts } w] \leq \varepsilon$

A prob. time TM is an NTM where each non-deterministic step is called a coin-flip step and has two legal moves. The probability of a branch b is 2^{-k}, where k is the # of coin flips that occurred on that branch.

$$BPP = \{A \mid \exists \text{prob. time TM that decides } A \text{ with error probability} \leq 1/3\}$$

Amplification lemma: Says that A probabilistic poly-time TM with error probability $\varepsilon < \frac{1}{2}$ has another equivalent probabilistic poly time TM with error ε' as small as you want.

Problems in ***BPP***	*Details*
PRIMES and *COMPOSITES*	Use Fermat's little theorem: p is prime and $a < p \Rightarrow \forall a, a^{p-1} \equiv 1 \pmod{p}$. If we can find an a such that the prime test is failed, then p is not prime. The good news is that most a will fail the test! BPP algorithm just picks two such $a's$ and accepts its input as *composite* if both a's fail the test. $\Pr[COMPOSITES \text{ accepts prime } w] = 0$ and . $\Pr[COMPOSITES \text{ accepts composite } w] = \frac{3}{4}$ (see Carmichael numbers)
EQ_{ROBP}	It's easy to build two ROBPs that agree on all but one or two inputs, so we have to use *the arithmetization trick* to turn two ROBPs that disagree rarely, into two ROBPs that disagree very often, while still keeping them equal if they were equal.

Definition: RP: always reject w when $w \notin A$

- $w \in A \Rightarrow \Pr[M \text{ accepts } w] \geq \frac{1}{2}$
- $w \notin A \Rightarrow \Pr[M \text{ accepts } w] = 0 \Leftrightarrow \Pr[M \text{ rejects } w] = 1$

Definition: $coRP$: always accept w when $w \in A$

- $w \in A \Rightarrow \Pr[M \text{ accepts } w] = 1 \Leftrightarrow \Pr[M \text{ rejects } w] = 0$
- $w \notin A \Rightarrow \Pr[M \text{ accepts } w] \leq \frac{1}{2}$

Problems in ***RP***	*Details*
COMPOSITES	$\Pr[COMPOSITES \text{ accepts prime } w] = 0$ and $\Pr[COMPOSITES \text{ accepts composite } w] \geq \frac{3}{4}$

Problems in ***coRP***	*Details*
PRIMES	$\Pr[PRIMES \text{ accepts prime } w] = 1$ and $\Pr[PRIMES \text{ accepts composite } w] \leq \frac{1}{4}$
EQ_{ROBP}	

Open problems

- GREEN means the statement is believed to be true, like $P \neq NP$ is believed to be true.
- BLACK means I have not investigated ☺

Open problems	*Reason, implications*
$P \neq NP$	If $P = NP$, that would imply $NP = coNP$, because $P = coP$ and $coP = coNP$.
$P \neq PSPACE$	
$NP \neq coNP$	
$TQBF \in NP$	It would imply $NP = PSPACE$.
$L \neq NL$	Savitch's theorem does not help here because $f(n) = \log n$ and the square of $\log n$ is not $\log n$.
$L \neq P$	
$NP \neq PSPACE$	
$NP \subseteq BPP$	