1. True/False/Open?

(a) i. \( L \neq P \)  
ii. \( P \neq \text{PSPACE} \)  
iii. \( L \neq P \lor P \neq \text{PSPACE} \)  
iv. \( P \neq \text{NP} \)  
v. \( \text{BPP} \neq \text{EXP} \)  
vii. \( L = P \implies \text{PSPACE} = \text{EXP} \)  
viii. \( \text{NP} \neq \text{coNP} \implies P \neq \text{NP} \)

(b) i. \( \text{NP} \)-complete languages are closed under \( \cup \)  
ii. \( \text{NP} \)-complete languages are closed under \( \cap \)  
iii. \( \text{NP} \)-complete languages are closed under \( \circ \)  
iv. \( \text{NP} \)-complete languages are closed under \( * \)  
v. \( \text{NP} \)-complete languages are closed under \( \vdash \)  
vi. \( \text{NP} \)-complete languages are closed under \( \setminus \)  
vii. \( \text{NP} \)-complete languages are closed under \( \mathcal{R} \)  
viii. \( \text{NP} \)-complete languages are closed under \( \cap \text{ REG} \)  

(c) i. \( \text{pSAT} = L \)  
ii. \( \text{pSAT} = \text{NL} \)  
iii. \( \text{pSAT} = P \)  
iv. \( \text{pSAT} = \text{BPP} \)  
v. \( \text{pSAT} = \text{NP} \)  
vii. \( \text{pSAT} = \text{PSPACE} \)  
viii. \( \text{pSAT} = \text{EXPSPACE} \)

Solutions

(a) vii. We already know that \( \text{PSPACE} \subseteq \text{EXP} \). Thus, it suffices to prove that \( \text{EXP} \subseteq \text{PSPACE} \). Consider an arbitrary language \( L \in \text{EXP} \). We will prove that \( L \in \text{PSPACE} \). Note that this would complete the proof. Since \( L \in \text{EXP} \), there exists a deterministic Turing machine, \( M_L \) which runs on inputs \( x \) of length \( n \) for time \( k \cdot 2^n^c \) for some fixed constants \( k, c \) and decides whether \( x \in L \). We construct a new language, \( L' \), as follows:

\[
L' = \{ x \# 0^{k \cdot 2^{|x|^c} - |x| - 1} : x \in L \}
\]

We think of \( L' \) as the padded version of \( L \). What was have done is constructed padded language where the input sizes have been artificially bloated. The idea is that the input \( x \), that was only \( |x| = n \) long, has been turned into the input \( x \# 0^{k \cdot 2^{|x|^c} - |x| - 1} \) which is now \( k \cdot 2^n^c \) long\(^2\). We now claim that \( L' \in \text{P} \). Consider the

---

\(^1\text{P} = \text{NP} \implies P = \text{BPP} \)

\(^2\text{This spirit would explain how to do away with any corner cases.}\)
An important pair of observations that will be used in the solutions to this part is that \( \emptyset \neq \Sigma^* \).

**L.** Let \( L \), \( L' \), \( L_0 \), and \( L_1 \) be \( \Sigma^* \)-complete languages. Consider the following deterministic Turing machine \( M_L \) for \( L' \): given its input, it first checks that it is well-formed, that is, it contains some string \( x \) followed by a \( \# \) and a string of 0s of length \( k \cdot 2^{|x|^c} - |x| - 1 \). If not, it rejects. If yes, it proceeds to remove the \( \# \) and 0s and then runs \( M_L \) on \( x \) and outputs whatever \( M_L \) outputs. Firstly, notice that just by definition, \( L(M_L') = L' \). Secondly, \( M_L' \) on an input of length \( N = k \cdot 2^{|x|^c} \), runs for \( O(N) \) time – this is because, that is the amount of time it takes to make sure the input is well-formed as well as the time taken by \( M_L \). This implies that \( L' \in \mathbb{P} \). Since \( L = \mathbb{P} \), this implies that \( L' \in \mathbb{L} \). This means that there is another deterministic \( \text{log space} \) machine, \( N_L' \), that decides \( L' \). The last step of the proof is to design a deterministic \( \text{polynomial space} \) Turing machine for \( L \) using \( N_L' \).

Consider the following deterministic Turing machine \( N_L \) for \( L \): given ints input, \( x \), it *implicitly* pads it with \( \#0^{k\cdot2^{|x|^c}-|x|-1} \). Note that it cannot actually do this since we want it to work in polynomial space. Then, it runs \( N_L' \) on the implicitly padded string. This means, that any time \( N_L' \) tries to read a symbol whose index is larger than \( |x| \), \( N_L \) communicates to \( N_L' \) that the symbol read was in fact 0 or \( \# \) depending on whether the index is greater than \( |x| + 1 \), or equal to \( |x| + 1 \). Another detail that is implicit in this is that \( N_L' \) never modifies the input. This is without loss of generality as \( N_L' \) is a \( \text{log space} \) machine and thus works in a model where the input tape is read-only. Again, by definition, \( L(N_L) = L \). Furthermore, \( N_L \) runs in \( O(\log 2^{|x|^c}) \), that is, polynomial space - this is because, that is the amount of space taken by \( N_L' \). This shows that \( L \in \text{PSPACE} \), completing the proof.

(b) An important pair of observations that will be used in the solutions to this part is that neither \( \emptyset \) nor \( \Sigma^* \) are \( \text{NP-complete} \). Thus is because \( \emptyset \nleq_p \Sigma^* \) and \( \Sigma^* \nleq_p \emptyset \).

i. Let \( L \) be a \( \text{NP-complete} \) language. Consider the languages \( L_0, L_1 \) defined by

\[
L_0 = \{0x : x \in L\} \cup \{1y : y \in \Sigma^*\} \cup \{\epsilon\}
\]

and

\[
L_1 = \{1x : x \in L\} \cup \{0y : y \in \Sigma^*\} \cup \{\epsilon\}
\]

It is easy to show that \( L_0, L_1 \) are \( \text{NP-complete} \) languages. However, \( L_0 \cup L_1 = \Sigma^* \) is not \( \text{NP-complete} \).

ii. Let \( L \) be a \( \text{NP-complete} \) language. Consider the languages \( L_0, L_1 \) defined by

\[
L_0 = \{0x : x \in L\}
\]

and

\[
L_1 = \{1x : x \in L\}
\]

It is trivial to see that \( L_0, L_1 \) are \( \text{NP-complete} \) languages. However, \( L_0 \cap L_1 = \emptyset \) is not \( \text{NP-complete} \).

iii. Let \( L \) be a \( \text{NP-complete} \) language. Consider the languages \( L_0, L_1 \) defined by

\[
L_0 = \{0x : x \in L\} \cup \{1y : y \in \Sigma^*\} \cup \{\epsilon\}
\]

and

\[
L_1 = \{1x : x \in L\} \cup \{0y : y \in \Sigma^*\} \cup \{\epsilon\}
\]

It is easy to show that \( L_0, L_1 \) are \( \text{NP-complete} \) languages. However, \( L_0 \circ L_1 = \Sigma^* \) is not \( \text{NP-complete} \).
iv. Let $L$ be an NP-complete language. Consider the language $L' = L \cup \{0, 1\}$. It is trivial to see that $L'$ is an NP-complete language. However, $L^* = \Sigma^*$ is not NP-complete.

vi. Let $L$ be an NP-complete language. However, $L \setminus L = \emptyset^3$ is not NP-complete.

viii. Let $L$ be an NP-complete language. Note that $\emptyset \in \text{REG}$. However, $L \cap \emptyset = \emptyset$ is not NP-complete.

(c) In this problem, for parts i. through vii., which are open, we will present two plausible scenarios under which the problem presented will be resolved in different ways, thus leaving the problem open for now.

i. **Scenario A:** If $L = \text{NP}$ (if $L = P$ and $P = \text{NP}$), then $P^{\text{SAT}} = L$. Furthermore, this is a plausible scenario as this is not known to imply that $P = \text{PSPACE}$ (which would in conjunction with $L = P$ imply $L = \text{PSPACE}$, contradicting the space-hierarchy theorem).

**Scenario B:** If $L \neq P$ (if $L \subsetneq P$), then $P^{\text{SAT}} \neq L$.

ii. **Scenario A:** If $NL = \text{NP}$ (if $NL = P$ and $P = \text{NP}$), then $P^{\text{SAT}} = NL$.

**Scenario B:** If $NL \neq P$ (if $NL \subsetneq P$), then $P^{\text{SAT}} \neq NL$.

iii. **Scenario A:** If $P = \text{NP}$, then $P^{\text{SAT}} = P$.

**Scenario B:** If $P \neq \text{NP}$, then $P^{\text{SAT}} \neq P$.

iv. **Scenario A:** If $P = \text{PSPACE}$, then $P^{\text{SAT}} = \text{BPP}$ since $P \subseteq P^{\text{SAT}} \subseteq \text{PSPACE}$ and $P \subseteq \text{BPP} \subseteq \text{PSPACE}$.

**Scenario B:** If $P = \text{BPP}$ and $P \neq \text{NP}$, then $P^{\text{SAT}} \neq \text{BPP}$.

v. Recall from the Sample Final Exam Problem 1 (i) that if $P^{\text{SAT}} \subseteq \text{NP} \cup \text{coNP}$, then $\text{NP} = \text{coNP}$.

**Scenario A:** If $P = \text{NP}$, then $P^{\text{SAT}} = \text{NP}$.

**Scenario B:** If $P \neq \text{NP}$, then $P^{\text{SAT}} \neq \text{NP}$.

vi. Recall from the Sample Final Exam Problem 1 (i) that if $P^{\text{SAT}} \subseteq \text{NP} \cup \text{coNP}$, then $\text{NP} = \text{coNP}$.

**Scenario A:** If $P = \text{NP}$, then $P^{\text{SAT}} = \text{coNP}$.

**Scenario B:** If $P \neq \text{NP}$, then $P^{\text{SAT}} \neq \text{coNP}$.

vii. **Scenario A:** If $P = \text{PSPACE}$, then $P^{\text{SAT}} = \text{PSPACE}$ since $P \subseteq P^{\text{SAT}} \subseteq \text{PSPACE}$.

**Scenario B:** If $P = \text{NP}$ and $P \neq \text{PSPACE}$, then $P^{\text{SAT}} \neq \text{PSPACE}$.

viii. $P^{\text{SAT}} \subseteq \text{PSPACE} \subseteq \text{EXPSPACE}$. If $P^{\text{SAT}} = \text{EXPSPACE}$, then $\text{PSPACE} = \text{EXPSPACE}$ which contradicts the space-hierarchy theorem.

2. Let

$$L_{\text{lin}} = \{M : L(M) \in \text{TIME}(n)\}$$

Show that $L_{\text{lin}}$ is undecidable.

**Solution.** We show that $A_{\text{TM}} \leq_m L_{\text{lin}}$ via a mapping reduction $f$ described as follows:

$f((M, w)) = \langle N \rangle$, where $N$ on input $x$:

(a) Check if $x \in \text{EQ}_{\text{REX}_1}$. If so, accept $x$ and halt. If not, continue.

(b) Run $M$ on $w$ and does whatever $M$ does.

---

3 It is very easy to construct NP-complete languages $L, L'$ such that $L \neq L'$ but $L \subseteq L'$ and thus $L \setminus L' = \emptyset$, if one desires a “non-trivial” counter-example.
Note that

- If $(M, w) \in A_{TM}$, then $L(N) = \Sigma^* \in \text{TIME}(n)$.
- If $(M, w) \not\in A_{TM}$, then $L(N) = \text{EQREX} \not\in \text{TIME}(n)$ as $\text{EQREX}$ is EXP-space complete.

3. Let

$$\text{BALANCED} = \{ w \in \{0, 1\}^* : \#_0(w) = \#_1(w) \}$$

Show that $\text{BALANCED} \in L$.

**Solution.** It is possible to determine $\#_0(w)$ and $\#_1(w)$ using two counters and then check if they are equal. This can clearly be done in log space.

4. Let $G$ represent a directed graph. How hard are the following problems?

$$\text{PATH} \leq = \{ \langle G, a, b, k \rangle : G \text{ contains a simple path of length at most } k \text{ from } a \text{ to } b \}$$

$$\text{PATH} \geq = \{ \langle G, a, b, k \rangle : G \text{ contains a simple path of length at least } k \text{ from } a \text{ to } b \}$$

**Solution.** $\text{PATH} \leq$ is NL-complete. Firstly, $\text{PATH} \leq \in \text{NL}$. This is easy to prove using the same argument that proves $\text{PATH} \in \text{NL}$. The only additional detail here is that our counter in this case counts only up to $k$ and not up to $n - 1$, where $n$ is the number of nodes in $G$. Secondly, $\text{PATH} \leq$ is NL-hard. We prove this by showing that $\text{PATH} \leq \ell \text{ PATH} \leq$. Our reduction simply maps $\langle G, s, t \rangle$ to $\langle G, s, t, n - 1 \rangle$. It is easy to see that this reduction can be performed in log space and that it is correct.

$\text{PATH} \geq$ is NP-complete. Firstly, $\text{PATH} \geq \in \text{NP}$. To see this, note that a certificate for $\langle G, s, t, k \rangle \in \text{PATH} \geq$ is a simple path of length at least $k$ from $a$ to $b$. This certificate is polynomially long (any simple path in $G$ can be at most $n - 1$ long, where $n$ is the number of nodes in $G$) and can be verified in polynomial time. Secondly, $\text{PATH} \geq$ is NP-hard. We prove this by showing that $\text{HAMPATH} \leq_p \text{PATH} \geq$. Our reduction simply maps $\langle G, s, t \rangle$ to $\langle G, s, t, n - 1 \rangle$. It is easy to see that this reduction can be performed in polynomial time and that it is correct.

5. Let $C$ be a class of languages. Show that if $\text{co}C \subseteq C$, then $\text{co}C = C$.

**Solution.** We need to show that $C \subseteq \text{co}C$. Consider an arbitrary $L \in C$. By definition, $\overline{L} \in \text{co}C$. Since $\text{co}C \subseteq C$, $\overline{L} \in C$. This implies that $L = \overline{L} \in \text{co}C$. This completes the proof.

6. Let

$$\text{SHORTEST-PATH} = \{ \langle G, k, s, t \rangle : \text{the shortest path from } s \text{ to } t \text{ in } G \text{ has length } k \}$$

Show that $\text{SHORTEST-PATH}$ is in NL. Also show that $\text{SHORTEST-PATH} \in L$ if and only if $L = \text{NL}$.

**Solution.** We will show that $\text{SHORTEST-PATH}$ is NL-complete. Notice that this solves the problem. Firstly, $\text{SHORTEST-PATH} \in \text{NL}$. Consider the language $\text{PATH} \leq$ defined in Problem 4. We have

$$\langle G, k, s, t \rangle \in \text{SHORTEST-PATH} \iff \langle G, s, t, k \rangle \in \text{PATH} \leq \land \langle G, s, t, k - 1 \rangle \in \overline{\text{PATH} \leq}$$

We already know that $\text{PATH} \leq \in \text{NL}$. Since $\text{NL} = \text{coNL}$, $\overline{\text{PATH} \leq} \in \text{NL}$. Combining these, we have $\text{SHORTEST-PATH} \in \text{NL}$ – the certificate for this problem would be a concatenation
of the certificates for PATH$_<$ and PATH$_{\leq}$. Secondly, SHORTEST-PATH is NL-hard. We prove this by showing that PATH$_{\leq} \leq_L$ SHORTEST-PATH. Given an instance $(G, s, t)$ of PATH, consider the layered version of $G$, namely, $G^{\text{layer}}$, defined as follows. Let $n$ denote the number of nodes in $G = (V, E)$. $G^{\text{layer}}$ consists of $n$ different copies of the nodes of $G$ whose edges move from one layer to the next. Formally, $G^{\text{layer}} = (V^{\text{layer}}, E^{\text{layer}})$, where

$$V^{\text{layer}} = \bigcup_{1 \leq i \leq n} V_i$$

with $V_i \cong V$ for each $1 \leq i \leq n$, and

$$E^{\text{layer}} = \{ (u_i, v_{i+1}) : (u, v) \in E \land 1 \leq i < n \} \cup \{ (u_i, u_{i+1}) : u \in V \land 1 \leq i < n \}$$

Now, consider the nodes $s_1$ and $t_n$. If there exists a path from $s_1$ to $t_n$, it must be of length exactly $n - 1$ as one would have to cross the layers from $V_1$ to $V_n$. Thus, the length of the shortest path from $s_1$ to $t_n$, if it exists, is $n - 1$. Furthermore, if and only if there exists a path from $s$ to $t$ in $G$, there exists a path from $s_1$ to $t_n$ in $G^{\text{layer}}$. Our reduction maps $(G, s, t)$ to $(G^{\text{layer}}, s_1, t_n, n - 1)$. We have just argued that this reduction is correct. It is easy to see that this reduction can be carried out in log space. This completes the proof.

7. Let

$$\text{COUNT-SAT} = \{ \langle \phi, k \rangle : \phi \text{ is a 3CNF that has precisely } k \text{ satisfying assignments} \}$$

Show that if $\text{COUNT-SAT} \in \text{NP}$ then $\text{NP} = \text{coNP}$.

**Solution.** We first show that $\text{3SAT} \leq_p \text{COUNT-SAT}$. The reduction simply maps a formula $\phi$ in 3CNF to the the pair $\langle \phi, 0 \rangle$. This can clearly be done in polynomial time. Furthermore, correctness follows by definition: $\phi \in \text{3SAT} \iff \phi$ has precisely 0 satisfying assignments. Thus, if $\text{COUNT-SAT} \in \text{NP}$, then $\text{3SAT} \in \text{NP}$. However, $\text{3SAT}$ is coNP-complete. Thus, this would imply that $\text{NP} = \text{coNP}$.

8. Show that $A_{\text{LBA}} \notin \text{NL}$.

**Solution.** We will show that $A_{\text{LBA}}$ is PSPACE-complete. Notice that this solves the problem as $\text{NL} \subseteq L^2$ by Savitch’s theorem and $L^2 \subseteq \text{PSPACE}$ by the space-hierarchy theorem. Firstly, $A_{\text{LBA}} \in \text{PSPACE}$. This is easy to see since we can simply simulate the linearly bounded automaton using linear space. Secondly, $A_{\text{LBA}}$ is PSPACE-hard. Consider an arbitrary language $L \in \text{PSPACE}$. We will prove that $L \leq_p A_{\text{LBA}}$ via a polynomial time reduction $f$. Note that this would complete the proof. Since $L \in \text{PSPACE}$, there exists a deterministic Turing machine, $M_L$ which runs on inputs $x$ of length $n$ using at most $k \cdot n^c$ space for some fixed constants $k, c$ and decides whether $x \in L$. The reduction $f$ maps $\langle x \rangle$ to $\langle M_L, x \cup^{k \cdot x^c - |x|} \rangle$. It is easy to see that this reduction can be performed in polynomial time. Furthermore, $M_L$’s behavior on $x \cup^{k \cdot x^c - |x|}$ will be that of an LBA since $M_L$ only uses $k \cdot |x|^c$ space on $x$. It is trivial$^4$ to see that the reduction is correct. This completes the proof.

9. Show that HALT$_{\text{TM}}$ is NP-hard.

**Solution.** We show that SAT $\leq_p$ HALT$_{\text{TM}}$ via a polynomial time reduction $f$ described as follows:

$$f(\langle \phi \rangle) = \langle N, \epsilon \rangle$$

where $N$ on input $x$:

---

$^4$Pedantic checks regarding delimiters, etc., can be easily done incorporated.
(a) Check if $\phi \in \text{SAT}$. If so, accept $x$ and halt. If not, continue.

(b) Loop

Note that

- If $\phi \in \text{SAT}$, then $(N, \epsilon) \in \text{HALT}_{TM}$ as $L(N) = \Sigma^*$.
- If $\phi \not\in \text{SAT}$, then $(N, \epsilon) \not\in \text{HALT}_{TM}$ as $L(N) = \emptyset$ and all rejections are by looping.

10. What is the relationship between $\text{coCFL}$ and $P$?

**Solution.** We would first like to prove a couple of general claims.

**Claim 1.** For any two classes of languages $C$ and $D$,

$$C \subseteq D \iff \text{co}C \subseteq \text{co}D$$

**Proof.** $\Rightarrow$: $C \subseteq D \implies \text{co}C \subseteq \text{co}D$

$$L \in \text{co}C \implies \overline{L} \in C$$

$$\implies \overline{L} \in D$$

$$\implies L \in \text{co}D$$

$\Leftarrow$: $\text{co}C \subseteq \text{co}D \implies C \subseteq D$

$$L \in C \implies \overline{L} \in \text{co}C$$

$$\implies \overline{L} \in \text{co}D$$

$$\implies L \in D$$

**Claim 2.** For any two classes of languages $C$ and $D$,

$$C = D \iff \text{co}C = \text{co}D$$

**Proof.** $\Rightarrow$: $C = D \implies \text{co}C = \text{co}D$

$$C = D \implies C \subseteq D \wedge D \subseteq C$$

$$\implies \text{co}C \subseteq \text{co}D \wedge \text{co}D \subseteq \text{co}C$$

$$\implies \text{co}C = \text{co}D$$

$\Leftarrow$: $\text{co}C = \text{co}D \implies C = D$

$$\text{co}C = \text{co}D \implies \text{co}C \subseteq \text{co}D \wedge \text{co}D \subseteq \text{co}C$$

$$\implies C \subseteq D \wedge D \subseteq C$$

$$\implies C = D$$

Now, we know that $\text{CFL} \subset P$. Therefore, $\text{coCFL} \subset P$ (P is closed under complement).
11. Let

\[ \text{MAX-CLIQUE} = \{ \langle G, k \rangle : k \text{ is the size of the largest clique in } G \} \]

Explain why the following argument fails to show that \( \text{MAX-CLIQUE} \in \text{coNP} \): To show that \( \langle G, k \rangle \notin \text{MAX-CLIQUE} \), it suffices to demonstrate the existence of a larger clique in \( G \) of size greater than \( k \), so the NP algorithm for \( \text{MAX-CLIQUE} \) just guesses the larger clique.

**Solution.** It may be the case that \( \langle G, k \rangle \notin \text{MAX-CLIQUE} \) since the largest clique in \( G \) is of size \( k' < k \).

12. Let

\[ \text{FACTOR} = \{ \langle n, m \rangle : \exists 1 < f < m \text{ such that } f \mid n \} \]

Show that \( \text{FACTOR} \in \text{NP} \cap \text{coNP} \).

**Solution.** To see that \( \text{FACTOR} \in \text{NP} \), notice that a certificate for \( \langle n, m \rangle \in \text{FACTOR} \) is the factor \( 1 < f < m \). This certificate is polynomially (in fact, linearly) long and only takes polynomial time to verify (division).

To see that \( \text{FACTOR} \in \text{coNP} \), notice that a certificate for \( \langle n, m \rangle \notin \text{FACTOR} \) is the prime factorization of \( n \). This certificate is polynomially long since a number can only have polynomially many distinct prime factors and the exponents corresponding to the prime factors can also be only polynomially long. This certificate can also be verified in polynomial time – we just check that the factors are prime\(^5\), that the factors raised to their exponents and multiplied yield \( n \) and that every factor is larger than \( m \).

13. Show that the following problems are NP-complete:

   (a) \( \text{DOUBLE-SAT} = \{ \phi : \#\phi \geq 2 \} \)

   (b) \( \frac{1}{3}\text{SAT} = \{ \phi : \phi \text{ in 3CNF has an assignment with exactly one true literal per clause} \} \)

   (c) \( \text{NFA-REJECT} = \{ \langle N, 1^n \rangle : N \text{ is an NFA that rejects some string of length } n \} \)

   (d) \( \text{PESKY-PATH} = \{ \langle G, s, t, \{ (u_i, v_i) \} \rangle : \exists \text{ s-t path in } G \text{ using exactly one of } (u_i, v_i) \forall i \} \)

**Solution.** We provide hints for coming up with reductions for NP-hardness of each problem.

(a) Reduce from \( \text{SAT} \). Given a formula \( \phi \), construct a new formula \( \phi' \) with an additional variable \( x \) such that \( \phi' = \phi \land (x \lor \overline{x}) \).

(b) Reduce from \( 3\text{SAT} \). Create, for every clause, 8 new variables \( x_{000}, x_{001}, \ldots, x_{111} \). The idea is that precisely 1 of these new variables should be true for each assignment of the clause.

(c) Reduce from \( 3\text{SAT} \). Consider the NFA you designed in Problem Set 5 Problem 2.

(d) Reduce from \( 3\text{SAT} \). Consider concatenating gadgets of the form described below.

\[ \begin{align*}
\text{variable} & \quad \text{clause } x_i \land x_j \land \overline{x_k} \\
\end{align*} \]

\(^5\)Primality testing can be done in polynomial time.