Problem 1

Show that a language can be decided by a TM (i.e., can be recognized by some TM that halts on all inputs), if and only if some machine can enumerate the language (i.e., enumerate all strings in this language) in lexicographic order. You may assume the alphabet is \{0, 1\}.

Consider \( L \) decidable by some TM \( M \). Say \( s_i \) is the \( i' \)th string in the lexicographical ordering of \( \Sigma^* \). We can then create an enumerator as follows:

\[
E = \text{"On any input} \]
\[
1. \text{For } i := 1, 2, \ldots
2. \text{Run } M \text{ on the input string } s_i
3. \text{If } M \text{ accepts, print } s_i. \text{ Else Continue.}
\]

This is a valid enumerator because it will never stay at any particular \( i \) forever (since \( M \) is a decider, it must halt and either accept or reject). Thus, if any \( s \) is in \( L \), then we will eventually reach \( s = s_k \) for some \( k \) in the lexicographical ordering, and eventually we will have \( i = k \).

Consider \( L \) that is enumerated in lexicographical order by some enumerator \( E \). WLOG assume that \(|L| \) is infinite, since finite languages are trivially decidable. We construct a decider for such \( L \) as follows:

\[
M = \text{"On input } w \]
\[
1. \text{Run } E \text{ until it prints out a string } s
2. \text{Accept if } s = w.
3. \text{Reject if } s > w \text{ in the lexicographical ordering.}
4. \text{Else Continue.}
\]
This decides the language since if \( w \in L \), eventually, \( E \) will print out \( w \) since there can only be a finite number of strings which are less than \( w \) lexicographically. Also, if \( w \notin L \), we will eventually print a string \( w' \) which is greater than \( w \) lexicographically, in which case we would reject (since \( E \) prints an infinite number of strings, since \( L \) is not finite WLOG).

**Problem 2**

Define the language \( C_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \) are two turing machines such that \( L(M_1) \subseteq L(M_2) \} \)

Show that \( C_{TM} \) is undecidable.

We reduce \( A_{TM} \) to \( C_{TM} \).

Assume the Turing Machine \( R \) decides \( C_{TM} \). We construct another Turing Machine \( S \) to decide \( A_{TM} \). Let the input to \( S \) be \( \langle M, \alpha \rangle \).

\( S \) works as follows. On input \( \langle M, \alpha \rangle \), \( S \) first writes down the description of another Turing machine \( M_1 \), such that \( M_1 \) only accepts the string \( \alpha \). That is, \( \forall x \in \{0, 1\}^* \), if \( x = \alpha \), then \( M_1 \) accepts; otherwise \( M_1 \) rejects. Now \( S \) computes \( R(\langle M_1, M \rangle) \) and \( S \) accepts if and only if \( R(\langle M_1, M \rangle) = 1 \) (i.e., \( R \) accepts the input \( \langle M, \alpha \rangle \)).

Since \( L(M_1) = \{\alpha\} \), we have that \( L(M_1) \subseteq L(M) \) if and only if \( M \) accepts \( \alpha \). Thus \( S \) can decide \( A_{TM} \) if \( R \) can decide \( C_{TM} \). Since \( A_{TM} \) is undecidable, \( C_{TM} \) is also undecidable.

**Problem 3**

Show that, if any language in \( coNP \) is \( NP \)-hard, then \( NP = coNP \).

Suppose a language \( L \in coNP \) is \( NP \)-hard. Then by definition \( \forall L' \in NP, L' \leq_p L \). Thus \( \forall L' \in NP \), we have \( L' \in coNP \) since \( L \in coNP \). This shows that \( NP \subseteq coNP \).

Similarly, \( \forall L' \in coNP \), we know that \( \bar{L}' \in NP \) by definition. Thus \( \bar{L}' \leq_p \bar{L} \). This implies that \( L' \leq_p \bar{L} \) by the same reduction. Since \( L \in coNP \) we have \( \bar{L} \in NP \). Thus we also have \( L' \in NP \). This shows that \( coNP \subseteq NP \).

Hence, \( NP = coNP \).
Problem 4

Let $\phi$ be a 3CNF. An $\neq$-assignment to the variables of $\phi$ is one where each clause contains two literals with unequal truth values.

(a) Show that any $\neq$-assignment automatically satisfies $\phi$, and the negation of any $\neq$-assignment to $\phi$ is also an $\neq$-assignment.

For any clause $C$ of $\phi$, the $\neq$-assignment will assign two literals in $C$ with different truth values. This means that one of them is 1, meaning $C$ is satisfied and thus $\phi$ is satisfied. For the negation, every literal takes its negation. Thus, for each clause, the literals which had different truth values originally will be flipped, but they will still have different truth values, so the negation is still a $\neq$-assignment.

(b) Let $\neq$-SAT be the collection of 3CNFs that have an $\neq$-assignment. Show that we obtain a polynomial time reduction from 3SAT to $\neq$-SAT by replacing each clause $c_i = (y_1 \lor y_2 \lor y_3)$ with the two clauses $$(y_1 \lor y_2 \lor z_i)$$ and $$(z_i \lor y_3 \lor b).$$ where $z_i$ is a new variable for each clause $c_i$ and $b$ is a single additional new variable.

Let $\phi$ be an input to the 3SAT problem. Use the replacements discussed to create $\phi'$ from $\phi$. It’s easy to see this transformation takes polynomial time since each clause only requires a constant number of operations. We now show that $\phi \in 3SAT$ if and only if $\phi' \in \neq$-SAT. Assuming that $\phi$ is satisfied by some assignment, we know that any clause $c_i = (y_1 \lor y_2 \lor y_3)$ must be 1 so $y_1 \lor y_2 \lor y_3 = 1$. To satisfy $(y_1 \lor y_2 \lor z_i)$ and $(\overline{z_i} \lor y_3 \lor b)$ we can set $z_i = (y_1 \lor y_2)$ and $b = 0$ (note that $b$ is set to 0 for all clauses). This is a $\neq$-assignment since $z_i = (y_1 \lor y_2)$ and $b = 0 = (\overline{z_i} \lor y_3)$, which by (a) automatically satisfies the two new clauses. Now assume $\phi'$ has a $\neq$-assignment. If $b = 1$ in this assignment, by (a) we take the negation of the assignment to get another $\neq$-assignment, which has $b = 0$. If $b = 0$ in the assignment, then we claim that the assignments to $y_1, \ldots, y_n$ give a satisfying assignment for $\phi$. To see this, note that for each clause $c_i$, the literals $y_1, y_2, y_3$ cannot be all 0, since if this happens then no $z_i$ can simultaneously satisfy $(y_1 \lor y_2 \lor z_i)$ and $(\overline{z_i} \lor y_3 \lor b)$. Thus $c_i$ is satisfied and $\phi$ is also satisfied. This shows that 3SAT reduces to $\neq$ SAT in polynomial time.
(c) Conclude that $\neq SAT$ is NP-complete.

Due to the reduction in part (b), we only need to show that $\neq SAT$ is in NP. For an input $\phi$, we can make a certificate $c$ to be an assignment to the variables. $\phi \in \neq SAT$ iff there exists a $\neq$ assignment, so we can just check $c$ to see if it is a $\neq$-assignment by checking if each clause has two literals of different values. This can be done in polynomial time, so $\neq SAT$ is in NP, and further $\neq SAT$ is NP-complete.