13.11 Show that for any solvable decision problem there is a way to encode instances of the problem so that the corresponding language can be recognized by a TM with linear time complexity.

Method One: Make The Encoding Solve The Problem

We define such an encoding, e, as follows

\[ e(x) = \begin{cases} 
  1x & \text{if } x \text{ is a yes-instance of the decision problem} \\
  0x & \text{otherwise} 
\end{cases} \]

The decision problem is solvable, so e is computable. e(x) can be recognized in a single move by examining the first character of the input. Linear time complexity means there are non-negative constants m, b such that \( \tau_T(|x|) \leq mx + b \) where \( \tau_T \) is the time complexity of the machine solving the decision problem. Clearly this is true for \( m = 0 \) and \( b = 1 \).

Method Two: Padding the input

For a given input x to a decision problem solved by the TM T, we can calculate \( \tau_T(|x|) \) by running T and counting the number of required steps. So we can define an encoding e such that \( e(x) = x\Delta^k \) such that \( |x\Delta^k| = \tau_T(|x|) \). Now for any input \( e(x') \), T requires exactly \( |e(x')| \) steps and so its time complexity is linear with respect to the encoded input.

14.4 a. Let \( L_1 \) and \( L_2 \) be languages over \( \Sigma_1 \) and \( \Sigma_2 \) respectively. Show that \( L_1 \leq_P L_2 \Rightarrow \overline{L_1} \leq_P \overline{L_2} \).

If \( L_1 \leq_P L_2 \) then there exists an \( f \) such that \( x \in L_1 \) if and only if \( f(x) \in L_2 \) and \( f \) can be computed in time polynomial in the length of \( x \). It follows from this definition that \( x \notin L_1 \) if and only if \( f(x) \notin L_2 \); and hence \( x \in \overline{L_1} \) if and only if \( f(x) \in \overline{L_2} \). Thus \( f \) is also a reduction from \( \overline{L_1} \) to \( \overline{L_2} \).

b. Let \( \text{coNP} = \{ \overline{L} | L \in \text{NP} \} \). Show that if there exists \( L \) such that \( L \) is \( \text{NP} \)-complete and \( \overline{L} \in \text{NP} \) then \( \text{coNP} \subseteq \text{NP} \).

First recall that the class \( \text{NP} \) is closed under polynomial-time reductions; that is, if \( L_1 \leq_P L_2 \) and \( L_2 \in \text{NP} \) then \( L_1 \in \text{NP} \).

\( L \) is \( \text{NP} \)-complete, so \( L \in \text{NP} \) and for all \( L_1 \in \text{NP} \), \( L_1 \leq_P L \). It follows from part a that for all \( L_2 \in \text{coNP}, L_2 \leq_P \overline{L} \). Since \( \overline{L} \in \text{NP} \) and \( \text{NP} \) is closed under polynomial-time reductions this implies that \( \text{coNP} \subseteq \text{NP} \).

14.5 Show that if \( L_1, L_2 \subseteq \Sigma^*, L_1 \in \text{P}, \) and \( L_2 \) is neither \( \emptyset \) nor \( \Sigma^* \), then \( L_1 \leq_P L_2 \).

By assumption, there exists \( x_1 \in L_2 \) and \( x_2 \notin L_2 \). Let \( f \), the function which carries out this reduction, be defined as

\[ f(x) = \begin{cases} 
  x_1 & \text{if } x \in L_1 \\
  x_2 & \text{otherwise.} 
\end{cases} \]

Note that membership in \( L_1 \) can be determined in polynomial time, so \( f \) can be computed in polynomial time. \( x \in L_1 \) if and only if \( f(x) = x_1 \in L_1 \), so \( L_1 \leq_P L_2 \).

14.6 a. If every instance of a problem \( P_1 \) is an instance of a problem \( P_2 \), and if \( P_2 \) is \( \text{NP} \)-hard, then \( P_1 \) is \( \text{NP} \)-hard. True or false?
False. $P_1$ could have no instances, and hence be trivial to solve (the algorithm could always answer “no”).

At the language level this is perhaps more clear: $L_1 = \emptyset \subseteq L_2$ for any $L_2$ but $\emptyset \in \mathcal{P}$. More generally, let $L'_2 \subseteq L_2$ be the hard instances of $L_2$; it could be that $L'_2 \cap L_1 = \emptyset$ and hence $L_1 \in \mathcal{P}$.

b. Show that $3\text{-SAT} \leq_p \text{CNF-SAT}$. Every instance of $3\text{-SAT}$ is an instance of CNF-SAT and $X \in 3\text{-SAT}$ if and only if $X \in \text{CNF-SAT}$, so the identity function is a polynomial reduction from $3\text{-SAT}$ to CNF-SAT.

c. Generalize part b in some way. For any two problems $P_1, P_2$ if all yes-instances of $P_1$ are yes-instances of $P_2$ and all no-instances of $P_1$ are no instances of $P_2$ then the identity function is a polynomial time reduction from $P_1$ to $P_2$.

14.9 Show that if $k \geq 4$ the $K\text{-SAT}$ problem is $\mathcal{NP}$-complete.

We will show this holds for $k \geq 3$ by induction.

Base Case: 3-SAT is shown to be $\mathcal{NP}$-complete in Theorem 14.6.

Inductive Case: Assuming $K\text{-SAT}$ is $\mathcal{NP}$-complete, we will show $(K+1)\text{-SAT}$ is also $\mathcal{NP}$-complete. First, note that $(K+1)\text{-SAT}$ is in $\mathcal{NP}$ since a non-deterministically generated truth assignment can be evaluated in polynomial time.

Now we will show that $(K+1)\text{-SAT}$ is $\mathcal{NP}$-hard by reducing $K\text{-SAT}$ to it, completing the proof of $\mathcal{NP}$-completeness.

$K\text{-SAT} \leq_p (K+1)\text{-SAT}$

1. Let $F$ be the function which carries out the reduction. We define $F$ as follows: given an expression

$$X = A_1 \land \ldots \land A_n$$

where each $A_i$ is a $k$-term disjunction,

$$F(X) = (x_{k+1} \lor A_1) \land (\overline{x}_{k+1} \lor A_1) \land \ldots \land (x_{k+1} \lor A_n) \land (\overline{x}_{k+1} \lor A_n).$$

2. $F$ creates two disjunctions for every term in $X$ so it is computable in time proportional to $2|X|$.

3. For any boolean expression $Z$ and any truth assignment of $x_1$,

$$(x \lor Z) \land (\overline{x} \lor Z) = (T \lor Z) \land (F \lor Z) = T \land Z = Z$$

Thus $F(X)$ is satisfiable if and only if $X$ is satisfiable.