Problem 1

Let $T \subseteq 0^* \cup 1^*$ and $g$ is a polynomial time computable function such that $\forall x, g(x) \in T$. Show that the image of $T$ (i.e., $\{y | (\exists x \in T)[g(x) = y]\}$) is a sparse set and the number of plausible (i.e., things that based on length are reachable and could in concept be written—yes, we know, each $g(x)$ actually outputs just one string, but that is not the point here) outputs of $g(x)$ is bounded by some polynomial $poly(|x|)$.

Solution: For each length $n$, $n \neq 0$, $T$ only has two strings $1^n$ and $0^n$. Therefore, $T$ contains polynomially many elements at each length. Thus, $T$ is sparse. Since $\forall x \in \Sigma^*, g(x) \in T$ and $T \subseteq \Sigma^*$, then the image of $T$ is a subset of $T$. Therefore, the image of $T$ is sparse. $g$ is polynomial time computable, then exists integer $k$ such that $\forall x[|g(x)| \leq |x|^k + k]$. Since for each length $n$, $n \neq 0$, there are two strings and for $n = 0$, there is only one string $\epsilon$, so the total number of possible strings of $g(x)$ is $\sum_{i=1}^{|x|+k} 2 + 1 = 2|x|^k + 2k + 1$. Therefore, $g(x)$ is bounded by some polynomial $poly(|x|)$.

Problem 2

The subset sum problem $SUM$ is this: given a set of integers and an integer $s$, does any non-empty subset sum to $s$, i.e.

$SUM = \{ \langle A, s \rangle | A \text{ is a set of integers, } \exists B \subseteq A \text{ and the sum of all elements in } B \text{ is } s \}$. 

1
Let \( a \in A \), prove that \( \langle A, s \rangle \in SUM \) if and only if \( \langle A - \{a\}, s \rangle \in SUM \) or \( \langle A - \{a\}, s - a \rangle \in SUM \). (Hint: the purpose of this problem is to help you understand the next problem.)

Solution:

Proof.

\( \langle A, s \rangle \in SUM \implies \langle A - \{a\}, s \rangle \in SUM \) or \( \langle A - \{a\}, s - a \rangle \in SUM \)

Since \( \langle A, s \rangle \in SUM \), then \( \exists B \subseteq A \) such that \( S_B = s \), where \( S_B \) denotes the sum of all elements in \( B \). If \( a \in B \), then \( S_{B-\{a\}} = S_B - a = s - a \). As \( B - \{a\} \subseteq A - \{a\} \), then \( \langle A - \{a\}, s - a \rangle \in SUM \). If \( a \notin B \), then \( B = B - \{a\} \). Thus \( S_{B-\{a\}} = S_B = s \). As \( B - \{a\} \subseteq A - \{a\} \), then \( \langle A - \{a\}, s \rangle \in SUM \).

\( \langle A, s \rangle \in SUM \iff \langle A - \{a\}, s \rangle \in SUM \) or \( \langle A - \{a\}, s - a \rangle \in SUM \)

If \( \langle A - \{a\}, s \rangle \in SUM \), as \( A - \{a\} \subseteq A \), then \( \langle A, s \rangle \in SUM \). If \( \langle A - \{a\}, s - a \rangle \in SUM \), then \( \exists B \subseteq A - \{a\} \) such that \( S_B = s - a \). Clearly, \( B \cup \{a\} \subseteq A \) and \( S_{B\cup\{a\}} = s - a + a = s \). Therefore, \( \langle A, s \rangle \in SUM \).

\[ \square \]

Problem 3

Show that, by using the tree pruning algorithm directly, \( SUM \in P \) if there is a tally set \( T \) such that \( SUM \leq_m^p T \). (Hint: choosing a subset for a set is similar with choosing a assignment for a boolean formula. Hint hint: according to our last problem, in the search of the subset (by using the tree of course), choosing \( a \) means we can convert our problem to \( \langle A - \{a\}, s - a \rangle \). Not choosing \( a \) means we can convert our problem to \( \langle A - \{a\}, s \rangle \).)

Solution: Let \( T \) be a tally set that is \( \leq_m^p \) hard for \( NP \). Let \( g \) be a deterministic polynomial-time function many-one reducing \( SUM \) to \( T \). For any input \( \{\langle A, s \rangle\} \), \(|A| = m\), we can construct the following algorithm,
**Stage 0:** $C = \{(A, s)\}$

**Stage $i$ ($1 \leq i \leq m$):** assuming that the collection at the end of stage $i - 1$ is the following collection of pairs: $\{(A_1, s_1), \ldots, (A_l, s_l)\}$

**Step 1:** Let $C$ be the collection

$\{(A_1 - a_i, s_1 - a_i), (A_2 - a_i, s_2 - a_i), \ldots, (A_l - a_i, s_l - a_i), (A_1 - a_i, s_1), (A_2 - a_i, s_2), \ldots, (A_l - a_i, s_l)\}$

**Step 2:** Set $C'$ to be $\emptyset$

**Step 3:** For each pair $p$ in $C$ do:

1. Compute $g(p)$.
2. If $g(p) \in 1^*$ and for no formula $q \in C'$ does $g(p) = g(q)$, then add $p$ to $C'$

**End Stage $i$:** [C' is the collection that gets passed on to Stage $i + 1$]

**Stage $m + 1$:** $(A, s) \in SUM$ if $\exists (A_i, s_i) \in C$ such that $s_i = 0$.

According to Problem 2, we know Step 1 of Stage $i$ does no damage to our invariant. For $g(p) \notin 1^*$, $g(p) \notin T \implies p \notin SUM$. Thus, it is safe for us to drop a pair $p$ in this case. For any two pairs $p, q$, $p \in SUM \iff g(p) \in T$ and $q \in SUM \iff g(q)$. Then, if $g(p) = g(q)$, we know $p \in SUM \iff q \in SUM$. Thus, there is no need for us to keep both $p$ and $q$ in the set. Therefore, Step 2 and Step 3 do not damage our invariant. Since $g$ is polynomial time computable, then the length of its output is bounded by a polynomial. For $g(p) \in T$, we know the number of possible outputs is bounded by a polynomial. Therefore, the number of pairs we keep in the set in each stage is bounded by a polynomial. There are only $m + 2$ stages in the algorithm. Therefore, this algorithm is polynomial-time computable.

**Problem 4**

Explain why both algorithms of Theorem 1.2 and 1.4 can be computed in polynomial time and by deterministic Turing machine and try to prove Corollary 1.5.
Solution: Both algorithms for Theorem 1.2 and 1.4 is deterministic in each step since we have not guessed something by using nondeterministic Turing machine. Therefore, both algorithms are deterministic.

(1) For theorem 1.2, by following the algorithm of theorem, we know for each stage, the number of formulas is bounded by the number of strings in the Tally set and this is also the reason for us to take the tree-pruning operation. And the number of strings in Tally set is polynomial and the reduction from SAT to Tally set we construct is also polynomial. Therefore, our algorithm is polynomial-time.

(2) For theorem 1.4, there are two situations.

- First, if this algorithm never meets the requirement of step 4 in any stage \( i \), that means in each stage, there are a limited number of formulas and the number is bounded by a polynomial \( p_d(p_k(|F|)) \). By considering all stages in the algorithm, we can obtain a maximum of number of formulas in a stage, named \( p_{max} = \max p_d(p_k(|F|)) \). We summarize the number of formulas in all stages and the number of whole formulas is still bounded by polynomial. And the computation time is bounded by polynomial time. Under this situation, the algorithm is polynomial-time.

- Then, if the algorithm steps into the step 4, this algorithm will go to the end and give the answer \( F \in SAT \) directly. And this algorithm consumes time even less than the first situation. Under this situation, the algorithm is still polynomial-time.

Proof. Recall the corollary 1.5, if there is a sparse coNP-complete set, then P=NP. By the theorem 1.4, we have already known that if there is a sparse set that is \( \leq^p_m \)-hard for coNP, then P=NP. coNP-complete means a language \( L \in coNP \) and for all languages \( L' \in coNP \), \( L' \leq^p_m L \), then \( L \) is coNP-complete. Therefore, coNP-complete is a subset of coNP-hard. Since theorem 1.4 holds, corollary 1.5 is also true.

Problem 5

By following the proof of Theorem 1.4, we stop the algorithm and say \( F \in SAT \) if step 4 is ever invoked. Please design an deterministic polynomial-time algorithm to find at least one possible assignment for \( F \) and say \( F \in SAT \) under the same assumption of Theorem 1.4 and explain it.
Solution: We can obtain at least one possible assignment if $F \in SAT$ by just modifying a little of this procedure of algorithm for theorem 1.4.

In the original algorithm, we stop and say $F \in SAT$ when step 4 is invoked. However, if we stop here we just know that $F \in SAT$, we can not make sure what is the possible assignment of formula $F$. So we change step 4 to the operation below.

**Step 4(new)** If $C'$ contains at least $p_{d}(p_{k}(|F|)) + 1$ elements, we keep these boolean formulas in the set $C'$ and pass $C'$ to the next stage $i + 1$ directly. By this modification, we guarantee that there is at least one possible satisfiable assignment for $F$ left in $C'$ and for each stage, the number of boolean formulas is still polynomial and in the last stage, we still obtained a polynomial number of boolean formulas. So this algorithm is polynomial-time and can be computed by a deterministic Turing machine.