1. Let $\Gamma = \{0, 1, \sqcup\}$ be the tape alphabet of all TMs in this problem. Define the busy beaver function $BB : \mathbb{N} \to \mathbb{N}$ as follows. For each value of $k$, consider all $k$-state TMs that halt when started with a blank tape. Let $BB(k)$ be the maximum number of 1s that remain on the tape among all of these machines. Show that $BB$ is not a computable function.

**Answer:** Suppose on the contrary that $BB$ is computable. Then there exists a TM $F$ that computes $BB$. Without loss of generality, $F$ be a TM that, on input $1^n$, and halts with $1^{BB(n)}$ on the tape for all $n$. Now, we construct a TM $M$ that halts when started with a blank tape based on $F$: In Step 1, $M$ writes $n$ 1s on the tape. In Step 2, $M$ doubles the 1s in the tape. In Step 3, $M$ simulates $F$ (on the input string $1^{2n}$). Thus, $M$ will always halt with $BB(2n)$ 1s when it is started with a blank tape.

To implement $M$, we require at most $n$ states to perform Step 1, and a total of $c$ states to perform Steps 2 and 3, for some constant $c$. By definition, we have $BB(n+c) = \text{maximum number of 1s that a } (n+c)\text{-state TMs will halt with, which is at least the number of 1s that } M \text{ halts with. This implies } BB(n+c) \geq BB(2n) \text{ holds for all } n. \text{ However, it is easy to check that } BB(k) \text{ is a strictly increasing function (why?). Thus, } BB(n+c) < BB(2n) \text{ when } n > c, \text{ and we arrive at a contradiction.}

In conclusion, $BB(k)$ is not a computable function.

2. Let $\text{AMBIG}_{\text{CFG}} = \{G \mid G \text{ is an ambiguous CFG} \}$. Show that $\text{AMBIG}_{\text{CFG}}$ is undecidable. (Hint: Use a reduction from PCP. Given an instance $P = \left\{ \left[ \begin{array}{c} t_1 \\ b_1 \end{array} \right], \left[ \begin{array}{c} t_2 \\ b_2 \end{array} \right], \ldots, \left[ \begin{array}{c} t_k \\ b_k \end{array} \right] \right\}$, of the Post Correspondence Problem, construct a CFG $G$ with the rules

\[
S \to T \mid B \\
T \to t_1Ta_1 \mid \cdots \mid t_kTa_k \mid t_1a_1 \mid \cdots \mid t_ka_k \\
B \to b_1Ba_1 \mid \cdots \mid b_kBa_k \mid b_1a_1 \mid \cdots \mid b_ka_k,
\]

where $a_1, \ldots, a_k$ are new terminal symbols. Prove that this reduction work.)

**Answer:** (1) If $P$ has a match with $t_{i_1}t_{i_2}\cdots t_{i_t} = b_{i_1}b_{i_2}\cdots b_{i_t}$, then we observe that the string $t_{i_1}t_{i_2}\cdots t_{i_t}a_{i_r} \cdots a_{i_s}a_{i_t}$ has at least two derivations, one from $T$ and one from $B$. (2) If the CFG $G$ is ambiguous, some string $s$ has multiple derivations. As $s$ is generated from $G$, it can be written as $w a_{j_1}a_{j_2} \cdots a_{j_m}$ for some $w$ that does not contain symbols from $a_i$'s. By checking the grammar $G$, we observe that the derivation of $T$ and the derivation of $B$ can each generate at most one strings of the same form as $s$. In particular, the multiple derivations (actually, 2 derivations) of $s$ must be as follows:

\[
S \Rightarrow T \Rightarrow s = t_{j_m}t_{j_{m-1}} \cdots t_{j_1}a_{j_1}a_{j_2} \cdots a_{j_m} \\
S \Rightarrow B \Rightarrow s = b_{j_m}b_{j_{m-1}} \cdots b_{j_1}a_{j_1}a_{j_2} \cdots a_{j_m}
\]
Thus, \( t_j, t_{j-1}, \ldots t_1 = b_j, b_{j-1}, \ldots b_1 \) and we can get a match of \( P \).

Combining (1) and (2), we show that \( P \) has a match if and only if \( G \) is ambiguous. Thus, the reduction from PCP to \( \text{AMBIG}_{\text{CFG}} \) works, and \( \text{AMBIG}_{\text{CFG}} \) is undecidable.

3. Define a two-headed finite automaton (2DFA) to be a deterministic finite automaton that has two read-only, bidirectional heads that start at the left-hand end of the input tape and can be independently controlled to move in either direction. The tape of a 2DFA is finite and is just large enough to contain the input plus two additional blank tape cells, one on the left-hand end and one on the right-hand end, that serve as delimiters. A 2DFA accepts its input by entering a special accept state. For example, a 2DFA can recognize the language \( \{a^n b^n c^n \mid n \geq 0\} \).

(a) Let \( A_{2\text{DFA}} = \{\langle M, x \rangle \mid M \text{ is a 2DFA and } M \text{ accepts } x \} \). Show that \( A_{2\text{DFA}} \) is decidable.

(b) Let \( E_{2\text{DFA}} = \{\langle M \rangle \mid M \text{ is a 2DFA and } L(M) = \{\} \} \). Show that \( E_{2\text{DFA}} \) is not decidable.

**Answer:**

(a) Let \( M \) be a 2DFA \( M \) and let \( s \) be the number of states in \( M \). On input string \( x \), there are at most \( s(|x| + 2)^2 \) distinct configurations for \( M \). Thus, to decide if \( M \) accepts \( x \), we can simulate \( M \) for at most \( s(|x| + 2)^2 \) steps and get the answer (why?).

(b) Suppose on the contrary that \( E_{2\text{DFA}} \) is decidable. Let \( D \) be a DTM that decides \( E_{2\text{DFA}} \). We now construct a TM \( E \) based on \( D \) for deciding \( E_{\text{TM}} \) as follows:

\[ M = \langle M \rangle \]

1. Construct a 2DFA \( M' \) which recognizes the accepting computation history of \( M \).
   (How to can a 2DFA \( M' \) perform this task?)

2. Run \( D \) on \( M' \). If \( D \) accepts, accept. Otherwise, reject."

Since \( E_{\text{TM}} \) is undecidable, contradiction occurs. In conclusion, \( E_{2\text{DFA}} \) is thus undecidable.

4. Let \( J = \{w \mid \text{either } w = 0x \text{ for some } x \in A_T, \text{ or } w = 1y \text{ for some } y \notin A_T \} \). Show that neither \( J \) nor the complement of \( J \) is Turing-recognizable.

**Answer:** Let \( A \) be the language \( \{\langle M, x \rangle \mid M \text{ is a TM and } M \text{ does not accept } x \} \). It is easy to check that \( A \) is not Turing-recognizable (by reduction from \( \overline{A_{\text{TM}}} \)). We first show how to reduce \( A \) to \( J \). This is done by the reduction function \( f(w) = 1w \), so that \( w \) is in \( A \) if and only if \( f(w) \) is in \( J \). Obviously, the function \( f \) is computable. As \( A \) is not Turing-recognizable, \( J \) is not Turing-recognizable.

---

1Formally, \( M' \) shall accept an input string \( c_1 \#c_2\# \cdots \#c_k \) if it corresponds to an accepting computation history of \( M \) on some input string. To check it, \( M' \) checks if \( c_1 \) is a valid start configuration, checks if \( c_k \) is a valid accepting configuration, and checks if each \( c_i \) follows legally from \( c_{i-1} \). The first two steps can be done easily using one tape head.

The difficulty lies in the last step. When we want to check if \( c_i \) follows legally from \( c_{i-1} \), we need to compare the symbols in these two configurations sequentially. However, we cannot check the \( j \)th symbol in \( c_{i-1} \), then compare with the \( j \)th symbol in \( c_i \), and get back to the \((j+1)\)th symbol in \( c_{i-1} \) for the next comparison, since the tape is read-only so that we cannot put ‘markers’ on the tape. Fortunately, this problem can be solved if we are using two tape heads, so that the tape head are always on the symbols that we want to compare, and can advance together. In other words, we can construct a 2DFA to recognize accepting computation history for any TM \( M \).
We next show how to reduce \( \overline{A_{TM}} \) to \( J \). This is done by the reduction function \( g(w) = 0w \), so that \( w \) is in \( \overline{A_{TM}} \) if and only if \( g(w) \) is in \( J \). Again, the function \( g \) is computable. Since \( \overline{A_{TM}} \) is not Turing-recognizable, \( J \) is not Turing-recognizable.

5. **Rice’s theorem.** Let \( P \) be any nontrivial property of the language of a Turing machine. Prove that the problem of determining whether a given Turing machine’s language has property \( P \) is undecidable.

In more formal terms, let \( P \) be a language consisting of Turing machine descriptions where \( P \) fulfills two conditions. First, \( P \) is nontrivial—it contains some, but not all, TM descriptions. Second, \( P \) is a property of the TM’s language—whenever \( L(M_1) = L(M_2) \), we have \( \langle M_1 \rangle \in P \) if and only if \( \langle M_2 \rangle \in P \). Here, \( M_1 \) and \( M_2 \) are any TMs. Prove that \( P \) is an undecidable language.

**Answer:** Assume on the contrary that \( P \) is a decidable language satisfying the properties and let \( R_P \) be a TM that decides \( P \). We show how to decide \( A_{TM} \) using \( R_P \) by constructing TM \( S \). First, let \( T_\emptyset \) be a TM that always reject, so \( L(T_\emptyset) = \emptyset \). We may assume that \( \langle T_\emptyset \rangle \notin P \) without of generality, because we could proceed with \( \overline{P} \) instead of \( P \) if \( \langle T_\emptyset \rangle \in P \). Because \( P \) is non-trivial, there exists a TM \( T \) with \( \langle T \rangle \in P \). Then, we can construct \( S \) based on \( T \) and \( R_P \) as follows:

\[
S = \text{“On input } \langle M, w \rangle \text{:} \\
1. \text{Use } M \text{ and } w \text{ to construct the following TM } M_w. \\
   M_w = \text{“On input } x \text{:} \\
   1. \text{Simulate } M \text{ on } w. \text{ If it halts and rejects, reject.} \\
   2. \text{Simulate } T \text{ on } x. \text{ If } T \text{ accepts } x, \text{ accept.”} \\
2. \text{Use TM } R_P \text{ to determine if } \langle M_w \rangle \in P. \text{ If YES, accept. Else, reject.”}
\]

Note that TM \( M_w \) has the property that (1) if \( M \) accepts \( w \), \( L(M_w) = L(T) \), and (2) if \( M \) does not accept \( w \), \( L(M_w) = \emptyset = L(T_\emptyset) \). In other words, \( \langle M_w \rangle \in P \) if and only if \( M \) accepts \( w \).

Since the construction of \( M_w \) from \( T \), \( M \), and \( w \) takes finite steps, the TM \( S \) is a decider for \( A_{TM} \). This creates a contradiction since \( A_{TM} \) is a undecidable language. In conclusion, \( P \) is undecidable.