1. (Sipser problem 5.21, **20 points**) Let $AMBIG_{CFG} = \{[G] \mid G \text{ is an ambiguis CFG}\}$ (where $[G]$ denotes a string that represents the grammar). Show that $AMBIG_{CFG}$ is undecidable.
Hint: You can use a reduction from PCP. Given an instance

$$P = \left\{ \boxed{\frac{t_1}{b_1}}, \boxed{\frac{t_2}{b_2}}, \ldots \boxed{\frac{t_k}{b_k}} \right\},$$

of the Post Correspondence Problem, construct a CFG G with the rules

$$\begin{array}{lcl} S & \rightarrow & T \mid B \\ T & \rightarrow & t_1 T \mathsf{a}_1 \mid \ldots \mid t_k T \mathsf{a}_k \mid t_1 \mathsf{a}_1 \mid \ldots \mid t_k \mathsf{a}_k \\ B & \rightarrow & b_1 B \mathsf{a}_1 \mid \ldots \mid b_k B \mathsf{a}_k \mid b_1 \mathsf{a}_1 \mid \ldots \mid b_k \mathsf{a}_k \ , \end{array}$$

where $\mathsf{a}_1, \ldots, \mathsf{a}_k$ are new terminal symbols. Prove that this reduction works.

Solution:

If P is solvable, then G is ambiguous.
Let $i_1, i_2, \ldots i_n$ be a solution to P. Let

$$\begin{array}{rcl} w & = & t_{i_1} t_{i_2} \cdots t_{i_n} \mathsf{a}_{i_n} \cdots \mathsf{a}_{i_2} \mathsf{a}_{i_1} \\ & = & b_{i_1} b_{i_2} \cdots b_{i_n} \mathsf{a}_{i_n} \cdots \mathsf{a}_{i_2} \mathsf{a}_{i_1} \end{array}$$

The string w has two derivations:

$$\begin{array}{lcl} S & \stackrel{S \rightarrow T}{\longrightarrow} & T \\ & \stackrel{T \rightarrow t_{i_1} T \mathsf{a}_{i_1}}{\longrightarrow} & t_{i_1} T \mathsf{a}_{i_1} \\ & \stackrel{T \rightarrow t_{i_2} T \mathsf{a}_{i_2}}{\longrightarrow} & t_{i_1} t_{i_2} T \mathsf{a}_{i_2} \mathsf{a}_{i_1} \\ & \stackrel{T \rightarrow \cdots}{\longrightarrow} & t_{i_1} t_{i_2} \ldots T \cdots \mathsf{a}_{i_2} \mathsf{a}_{i_1} \\ & \stackrel{T \rightarrow t_{i_n} \mathsf{a}_{i_n}}{\longrightarrow} & t_{i_1} t_{i_2} \cdots t_{i_n} \mathsf{a}_{i_n} \cdots \mathsf{a}_{i_2} \mathsf{a}_{i_1} \end{array}$$

and

$$\begin{array}{lcl} S & \stackrel{S \rightarrow B}{\longrightarrow} & B \\ & \stackrel{B \rightarrow b_{i_1} B \mathsf{a}_{i_1}}{\longrightarrow} & b_{i_1} B \mathsf{a}_{i_1} \\ & \stackrel{B \rightarrow b_{i_2} B \mathsf{a}_{i_2}}{\longrightarrow} & b_{i_1} b_{i_2} B \mathsf{a}_{i_2} \mathsf{a}_{i_1} \\ & \stackrel{B \rightarrow \cdots}{\longrightarrow} & b_{i_1} b_{i_2} \ldots B \cdots \mathsf{a}_{i_2} \mathsf{a}_{i_1} \\ & \stackrel{B \rightarrow b_{i_n} \mathsf{a}_{i_n}}{\longrightarrow} & b_{i_1} b_{i_2} \cdots b_{i_n} \mathsf{a}_{i_n} \cdots \mathsf{a}_{i_2} \mathsf{a}_{i_1} \end{array}$$

Thus, G is ambibuous.

If G is ambiguous, then P is solvable.
Let G_T be the same grammar as G except that it has T as the start variable and likewise for G_B.

G_T is unambiguous.
This is because the string of a_i's at the end of any string derived from G_T describes the sequence of steps taken in the derivation. Thus, two strings can have the same suffix of a_i's iff they have the same derivation. This means that if G_T generates x and y and $x = y$, then x and y have the same derivation. Therefore, G_T is unambiguous.

G_B is unambiguous.

The proof is equivalent to that for G_T.

Thus, if w has two derivations in G, one must start with the rule $S \to T$ and the other with the rule $S \to B$. This means that there exist $i_1, \ldots i_m$ such that $w = t_{i_1} \cdots t_{i_m} \mathtt{a}_{i_m} \cdots \mathtt{a}_{i_1}$ and $j_1, \ldots j_n$ such that $w = b_{j_1} \cdots b_{j_n} \mathtt{a}_{j_n} \cdots \mathtt{a}_{j_1}$. Because the $\mathtt{a}_i$'s don't appear in any of the t's or b's, we have that $m = n$, $i_1 = j_1$, $i_2 = j_2$, … and $i_n = j_n$. This means that

$$t_{i_1} t_{i_2} \cdots t_{i_n} \quad = \quad b_{i_1} b_{i_2} \cdots b_{i_n}$$

This is a solution to P. Thus, P is solvable as required.

2. (Sipser problems 5.22 and 5.23, **20 points**)

(a) Show that A is Turing-recognizable iff $A \leq_m A_{TM}$.

Solution:

If $A \leq_m A_{TM}$, then A is Turing recognizable.

Assume that $A \leq_m A_{TM}$. I'll construct M_A, a TM that recognizes A. On input w, M_A uses the reduction for $A \leq_m A_{TM}$ to compute the description of a Turing machine M' and a string w' such that M' accepts w' iff $w \in A$. M_A then runs M'. If M' accepts (resp. rejects or loops), then M_A accepts (resp. rejects or loops). M_A accepts w iff $w \in A$; otherwise M_A rejects or loops. M_A is a TM that recognizes A, thus A is Turing-recognizable.

If A is Turing-recognizable, then $A \leq_m A_{TM}$.

Assume that A is Turing-recognizable. Thus, there is some TM that recognizes A. Let M_A be such a TM, and let $[M_A]$ denote the string that describes M_A. For any string, w, $w \in A$ iff $[M_A]\#w \in A_{TM}$. Clearly, the function that maps w to $[M_A]\#w$ is Turing computable. Thus, we've shown that $A \leq_m A_{TM}$ as required.

Note: these proofs are nearly trivial. They're just using the definitions of reduction, Turing-recognizable, and A_{TM}. The point behind this problem is to make sure that you understand what it means for one language to be Turing-reducible to another.

(b) Show that A is Turing-decidable iff $A \leq_m 0^*1^*$.

Solution:

If $A \leq_m 0^*1^*$, then A is Turing-decidable.

Assume that $A \leq_m 0^*1^*$. I'll construct M_A, a TM that decides A. On input w, M_A uses the reduction for $A \leq_m 0^*1^*$ to compute a new string, x such that $x \in 0^*1^*$ iff $w \in A$. M_A then uses its finite control to implement a DFA that determines whether or not $x \in 0^*1^*$. If $x \in 0^*1^*$, then M_A accepts; otherwise, M_A rejects. Note that the reduction step can't loop (by the definition of Turing-reducible) and the DFA step can't loop (because the DFA reads one symbol of x at each step and decides when it finishes reading x). Thus, M_A never loops. Therefore, M_A is a decider for A which means that A is Turing-decidable.

If A is Turing-decidable, then $A \leq_m 0^*1^*$.

Assume that A is Turing-decidable. Thus, there is some TM that decides A. Let M_A be such a TM. To reduce A to 0^*1^*, construct a Turing machine, M, that does the following on input w:

Run M_A on input w.

If M_A accepts w, erase the tape.

If M_A rejects w, erase the tape and write the string `10`.

M_A cannot loop, it's a decider.

Thus, if $w \in A$, then M writes ϵ on its tape, and $\epsilon \in 0^*1^*$. Otherwise, M writes `01` on its tape, and `01` $\notin 0^*1^*$. Thus, M reduces A to 0^*1^*.

Note: notice the cut-and-paste job I did to take the solution to part (a) and rewrite it into a solution for part (b). While these proofs are very simple, they give you the basic template for reduction proofs.

3. (Sipser problem 5.24, **20 points**) Let $J = \{w \mid \text{either } w = 0x \text{ for some } x \in A_{TM} \text{ or } w = 1y \text{ for some } y \notin A_{TM}\}$. Show that neither J nor $\overline{J}$ is Turing-recognizable.

   Solution:

   J is not Turing-recognizable.

   We reduce $\overline{A_{TM}}$ to J by constructing $M_{\overline{A_{TM}}}$, a TM that recognizes $\overline{A_{TM}}$ using M_J, a TM that recognizes J. On input $[M]\#w$, $M_{\overline{A_{TM}}}$ moves every input symbol on square to the right and writes a 0 on the leftmost square. It then moves its head to the leftmost square and runs M_J. If M_J accepts then $[M]\#w \notin A_{TM} \Leftrightarrow [M]\#w \in \overline{A_{TM}}$. We have reduced $\overline{A_{TM}}$ to J; $\overline{A_{TM}}$ is not Turing-recognizable, therefore, J is not Turing-recognizable either.

   $\overline{J}$ is not Turing-recognizable.

   We use basically the same construction as before, but this time we replace $[M]\#w$ with $0[M]\#w$.

4. (Sipser problem 5.34, **30 points**) Consider the problem of determing whether a PDA accepts some string of the form $\{ww \mid w \in \{0,1\}^*\}$. Use the computation history method to show that this problem is undecidable.

   Solution: Let h be a computational history. We can write h as

$$h \quad = \quad \# \, \mathit{config}_0 \, \# \, \mathit{config}_1^{\mathcal{R}} \, \# \, \mathit{config}_2 \, \# \, \mathit{config}_3^{\mathcal{R}} \, \# \, \cdots \, \mathit{config}_m \, \#$$

(where config_m is reversed if m is odd). Let M be at TM and w be a string. I'll now describe a PDA, P, that accepts a string of the form hh iff h describes a valid computational history for M accepting w.

Initially, P pushes each even numbered configuration onto its stack, and then pops each off while verifying the subsequent odd configuration. The $\#$ symbols separate successive configurations; so, P knows when to change from pushing to popping.

When P sees two consecutive $\#$ symbols, it skips the next configuration config_0. It then pushes each of the odd numbered confuration onto its stack, and then pops each off while verifying the subsequent even configuration.

In the course of these actions, P also checks that config_0 is the correct initial configuration for M running on input w and that config_m is an accepting configuration. The languages corresponding to these checks are regular, and P performs these checks using its finite state (see also HW 9, question 1).

P accepts a string of the form hh iff h is a valid computation history for M accepting w.

Note that P may accept strings that are not of the form hh whether or not M accepts h. In particular, one could run P on the input:

$$\# \, \mathit{config}_0 \, \# \, \mathit{config}_1^{\mathcal{R}} \, \# \, \mathit{config}_2 \, \#\# \, \mathit{config}_3 \, \# \, \mathit{config}_4 \, \#$$

where

- config_0 is the correct initial configuration for M running with input w.
- config_1 is the correct successor to config_0.
- config_2 is an arbitrary accepting configuration for M. Note that P does not verify that config_2 is a valid successor of config_1.
- config_3 is an arbitrary configuration. As described above, P does not verify that this is the correct initial configuration for M running with input w.
- config_4 is an arbitrary accepting configuration for M. P does not verify that config_4 is a valid successor of config_3.

This string is not of the form ww. P has no way to verify that its input is of the form ww.