Problem 5.21 (5 pts)
Let $AMBI_{CFG} = \{(G) \mid G \text{ is an ambiguous CFG}\}$. Show that $AMBI_{CFG}$ is undecidable.

(1) Let’s use the hint from the textbook and reduce PCP to $AMBI_{CFG}$. Given an instance $P$ of PCP, if $P$ has a match with $i_1, i_2, \ldots, i_m$ such that $t_{i_1}t_{i_2}\ldots t_{i_m} = b_{i_1}b_{i_2}\ldots b_{i_m}$, then we observe that the string $t_{i_1}t_{i_2}\ldots t_{i_k}a_{i_k}a_{i_2}a_{i_1}$ 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 $wa_{j_1}a_{j_2}\ldots a_{j_m}$ 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 $A$ can each generate at most one string of the same form as $s$. In particular, the multiple derivations (specifically, 2 derivations) of $s$ must be:

$S \Rightarrow T \Rightarrow^* s = t_{j_m}t_{j_{m-1}}\ldots t_{j_1}a_{j_1}a_{j_2}\ldots a_{j_m}$

$S \Rightarrow B \Rightarrow^* s = b_{j_m}b_{j_{m-1}}\ldots b_{j_1}a_{j_1}a_{j_2}\ldots a_{j_m}$

Therefore, $t_{j_m}t_{j_{m-1}}\ldots t_{j_1} = b_{j_m}b_{j_{m-1}}\ldots b_{j_1}$ and we can get a match of $P$.

Combining (1) and (2), we have shown that $P$ has a match if and only if $G$ is ambiguous. Thus, the reduction from PCP to $AMBI_{CFG}$ works, and $AMBI_{CFG}$ is undecidable.

Problem 5.23 (5 pts)
Show that $A$ is decidable iff $A \leq_m 0^*1^*$.

($\Rightarrow$) If $A \leq_m 0^*1^*$, then $A$ is decidable because $0^*1^*$ is a decidable language by Theorem 5.22.

($\Leftarrow$) If $A$ is decidable, then there exists some TM $R$ that decides $A$. That is, $R$ would receive an input $w$ and accept if $w$ is in $A$, reject if $w$ is not in $A$. To show $A \leq_m 0^*1^*$, we design a TM $Q$ that does the following: On input $w$, run $R$ on $w$. If $R$ accepts, outputs $01$; otherwise, outputs $10$. It is easy to check that:

$w \in A \iff \text{output of } Q \in 0^*1^*$

Thus, we obtain a mapping reduction of $A$ to $0^*1^*$.

Problem 5.24 (5 pts)
Let $J = \{w \mid \text{either } w = 0x \text{ for some } x \in A_{TM}, \text{or } w = 1y \text{ for some } y \in \overline{A_{TM}}\}$. Show that neither $J$ nor $\overline{J}$ is Turing-recognizable.

First, we show that $A_{TM} \leq_m J$. Design the follow TM $Q$: On input $\langle M, w \rangle$, write 0 followed by $\langle M, w \rangle$ in the tape and halts. It is easy to check that:

$\langle M, w \rangle \in A_{TM} \iff \text{output of } Q \in J$

Therefore, we obtain a mapping reduction of $A_{TM}$ to $J$.

Next, we must show that $A_{TM} \leq_m \overline{J}$. Design the following TM $R$: On input $\langle M, w \rangle$, write 1 follow by $\langle M, w \rangle$ in the tape and halts. It is easy to check that:

$\langle M, w \rangle \in \overline{A_{TM}} \iff \text{output of } R \in J$

Equivalently, we have:

$\langle M, w \rangle \in A_{TM} \iff \text{output of } R \in \overline{J}$

Thus, we obtain a mapping reduction of $A_{TM}$ to $\overline{J}$. 
Since \( A_{TM} \leq_m J \), we have \( \overline{A_{TM}} \leq_m \overline{J} \). This shows that \( \overline{J} \) is non-Turing-recognizable by Corollary 5.29 because \( A_{TM} \) is non-Turing-recognizable. Similarly, since \( A_{TM} \leq_m \overline{J} \), we have \( \overline{A_{TM}} \leq_m \overline{J} \). So, this shows that \( J \) is non-Turing-recognizable.

**Problem 5.33** (5 pts)
Consider the problem of determining 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.

We reduce \( A_{TM} \) to the given problem to prove it is undecidable. Given a string \( \langle M, w \rangle \) where \( M \) is a TM and \( x \) is an input to \( M \), we build a PDA that accepts a string of the form \( h\$h\$ \) iff \( x \in L(M) \). Here, \( h \) denotes the computation history of \( M \) on \( x \). Let us write the computation history of \( M \) on \( x \) as \( C_0C_1^R C_2C_3 ... C_n^R \), where \( C_0 \) is the initial configuration, \( C_n^R \) is the final accepting configuration (reversed if \( n \) is odd), even-indexed configurations are written in normal order, and odd-indexed configurations are reversed. (The question mark on \( C_n^R \) denotes that \( C_n \) may be written in reverse order depending on the parity of \( n \).)

Our PDA will push \( C_0 \) onto its stack while confirming that it is the initial configuration for \( M \) running with input \( w \). For each odd indexed configuration, the PDA pops the previous configuration off of its stack and verifies that the new configuration is the successor of the previous one. When the PDA reads the first \( S \), it skips configuration \( C_0 \) and then pushes \( C_1 \) onto the stack. This time, it reads the configurations confirming that each even-indexed configuration is the successor of the previous odd-indexed configurations pass and if the final configuration is an accepting one. Thus, this PDA accepts a string of the form \( ww \) iff \( M \) accepts \( x \), the string being \( h\$h\$ \).

Note that this PDA may accepts strings of the form \( y\$z\$ \) where \( y \neq z \) even if \( M \) does not accept \( x \). This is because our construction relies on the separate check that the input is of the form \( ww \) (that cannot be done by a PDA) to ensure that the two passes over the history are checking the same history.

**Problem 5.35** (10 pts)
Say that a variable \( A \) in CFG \( G \) is **necessary** if it appears in every derivation of some string \( w \in G \). Let \( NECESSARY_{CFG} = \{(G, A) \mid A \text{ is a necessary variable in } G \} \).

a. (5 pts) Show that \( NECESSARY_{CFG} \) is Turing-recognizable.

Let \( M \) be a TM that behaves as follows:

On input \( \langle G, A \rangle \):

1. Construct CFG \( G \setminus A \) by removing the nonterminal \( A \) and any production that mentions it from \( G \).
2. Enumerating strings \( w \) generated by \( G \). For each such string, simulate a decider for CFG \( A \) to test whether \( w \) is generated by \( G \setminus A \). If \( w \) is not generated by \( G \setminus A \) then accept, otherwise continue the search.
3. On other input, reject.

The language \( L(G \setminus A) \) consists of all and only those string \( w \in L(G) \) that have derivations that do not use \( A \). If \( w \in L(G) \) and \( w \) is not in \( L(G \setminus A) \), then \( A \) is necessary for \( G \) to generate \( w \in L(G) \). Eventually,
$M$ will identify such a $w$ and will accept. On the other hand, if $A$ is not necessary for $G$, then $L(G) = L(G \setminus A)$ and $M$ will loop. Thus, $M$ recognizes $\text{NECESSARY}_{CFG}$.

Note that it is not possible to prove this part by enumerating all strings $w \in L(G)$ and checking all derivations of each such string to see whether or not $A$ is used, because in general a string $w \in L(G)$ can have infinitely many derivations. So the checking procedure might never terminate.

b. (5 pts) Show that $\text{NECESSARY}_{CFG}$ is undecidable.

We will show $\text{ALL}_{CFG} \leq_m \text{NECESSARY}_{CFG}$.

The reduction $f$ is computed as follows:

On input $(G)$:
1. Construct $G_0$ by adding to $G$ a new nonterminal $A$, together with productions: $S \to A, A \to \epsilon$, and $A \to aA$ for each $a \in \Sigma$.
2. Output $(G_0, A)$.

Note that the grammar $G_0$ constructed by $f$ is always such that $L(G_0) = \Sigma^*$. Thus, if $L(G) = \Sigma^*$, then $A$ is not necessary for $G_0$, because every string $w \in \Sigma^*$ can already be derived from $G$, hence also from $G_0$ by a derivation not using $A$. Also, if $L(G) \neq \Sigma^*$, then $A$ is necessary for $G_0$. Because if $w \notin L(G)$ the $w$ can only be derived from $G_0$ by a derivation that uses $A$. To summarize, if $(G) \in \text{ALL}_{CFG}$, then $(G_0, A) \notin \text{NECESSARY}_{CFG}$, and if $(G) \notin \text{ALL}_{CFG}$, then $(G_0, A) \in \text{NECESSARY}_{CFG}$. Hence, $f$ is a reduction of $\text{ALL}_{CFG}$ to $\text{NECESSARY}_{CFG}$, as claimed.