Please note that there is more than one way to answer most of these questions. The following only represents a sample solution.

(1) 2.4 and 2.5 (b),(c),(e),(f): CFGs and PDAs

Solution:

(b) \( B = \{ w \mid w \text{ starts and ends with the same symbol} \} \)

2.4: We will assume that to start and end with the same symbol, the string must have at least one symbol and that a single symbol starts and ends with the same symbol.

\[
S \rightarrow 0A0 | 1A1 | 0 | 1 \\
A \rightarrow 0A | 1A | \epsilon
\]

2.5 Informal Description: We will nondeterministically guess if the string has only one symbol in which case we accept it without using the stack; otherwise, we push the first symbol read onto the stack. Then we will read every other symbol and nondeterministically guess if that is the last symbol read. If the last symbol read then matches the symbol on the stack and there is no more input we accept. Otherwise we reject.

(c) \( C = \{ w \mid \text{the length of } w \text{ is odd} \} \)

2.4:

\[
S \rightarrow 0A | 1A \\
A \rightarrow 0S | 1S | \epsilon
\]
2.5 Informal Description: The stack is not needed here at all. Therefore we will read the input and only accept if the length is odd, that is after the first symbol read or every other symbol read thereafter if it is the final symbol read.

(e) $E = \{w | w = w^R$, that is, $w$ is a palindrome$\}$

2.4:  
$S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \epsilon$

2.5 Informal Description: We begin by pushing the symbols read onto the stack. At each point we will nondeterministically guess if the middle of the string has been reached or if the next symbol read is the middle of the string and will not be put on the stack. Then we pop off the symbols from the stack if they match the input symbols read. If the symbols popped are exactly the same symbols that were pushed on earlier and the stack empties as the input is finished, then accept. Otherwise, reject.

(f) $F = \emptyset$, the emptyset

2.4:  
$S \rightarrow S$

Note: Since, no derivations terminate, the CFG cannot accept any strings, including the empty.

2.5 Informal Description: The PDA consists of one state that does not accept.
(2) 2.9 and 2.10: Ambiguous Grammar and PDA for $A = \{a^i b^j c^k | i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}$

Solution:

2.9:

$$
\begin{align*}
S & \rightarrow AB \mid CD \\
A & \rightarrow aAb \mid \epsilon \\
B & \rightarrow cB \mid \epsilon \\
C & \rightarrow aC \mid \epsilon \\
D & \rightarrow bDc \mid \epsilon
\end{align*}
$$

This grammar is ambiguous. To see this, note that $\epsilon \in A$ and $\epsilon$ can be derived with left derivations in two different ways. $S \Rightarrow AB \Rightarrow \epsilon B \Rightarrow \epsilon$ and also $S \Rightarrow CD \Rightarrow \epsilon D \Rightarrow \epsilon$. There are many other strings that this works for in the grammar above, but it suffices to show one. In fact, it can be shown that $A$ is inherently ambiguous as discussed on page 106 of Sipser, but we will not prove that here.

2.10 At the beginning nondeterministically break into two branches. In the first branch, for every $a$ read, push an $a$ on the stack. Nondeterministically guess when the first $b$ is read and begin popping symbols from the stack for each $b$ read. When the stack is empty, the only input remaining should be $c$s and then just read the input without adjusting the stack. If the symbols are read in the proper order ($a$s followed by $b$s followed by $c$s) and the stack is empty by the time the $b$s are done being read, then accept. Otherwise reject. In the second branch, for every $a$ read do not adjust the stack. Nondeterministically guess when the first $b$ is read and begin pushing symbols on the stack for each $b$ read. Nondeterministically guess when the first $c$ is read, begin popping symbols from the stack for each $c$ read. If the stack empties precisely when the input string is done and the symbols are read in the proper order ($a$s followed by $b$s followed by $c$s), accept. Otherwise, reject.

Below is a PDA state diagram that recognizes $A$, but is not required for you to submit on the homework.
2.16: For the following let \( A = (V_A, \Sigma_A, R_A, S_A) \) and \( B = (V_B, \Sigma_B, R_B, S_B) \) be CFGs with \( V_A \cap V_B = \emptyset \)

**Union:** Let \( C = (V_A \cup V_B \cup \{S\}, \Sigma_A \cup \Sigma_B, R_A \cup R_B \cup \{S \to S_A|S_B\}, S) \). Then \( L(C) = L(A) \cup L(B) \), since we, from the new start variable, we go to either the start variable of \( A \), which derives a string in \( A \), or to the start variable of \( B \) which derives a string in \( B \).

**Concatenation:** Let \( D = (V_A \cup V_B \cup \{S\}, \Sigma_A \cup \Sigma_B, R_A \cup R_B \cup \{S \to S_A S_B\}, S) \). Then \( L(D) = L(A) \circ L(B) \), since we from the new start variable we derive something in \( A \) followed by something in \( B \).

**Star:** Let \( E = (V_A \cup \{S\}, \Sigma_A, R_A \cup \{S \to S_A|\epsilon\}, S) \). Then \( L(E) = (L(A))^* \), since we from the new start variable we derive the concatenation of one or more strings in \( A \), or \( \epsilon \).

2.17: We will show that every regular language is context free. Let \( A \) be a regular language, then there exists a regular expression \( R \) such that \( L(R) = A \). We produce a CFG \( G \) such that \( L(G) = A \) as follows:

\[
G = grammar(R),
\]

where

\[
grammar ::= proc(R)
\]

If \( R = \emptyset \) then return \((\{S\}, \emptyset, \{S \to S\}, S)\), the grammar in 2.4 (f)

Else if \( R = \epsilon \) then return \((\{S\}, \emptyset, \{S \to \epsilon\}, S)\)

Else if \( R = a \) then return \((\{S\}, \{a\}, \{S \to a\}, S)\)

Else if \( R = (R_1 \cup R_2) \) then return the CFG for the union of \( grammar(R_1) \) and \( grammar(R_2) \) as specified in 2.16

Else if \( R = R_1 R_2 \) then return the CFG for the concatenation of \( grammar(R_1) \) and \( grammar(R_2) \) as specified in 2.16

Else if \( R = (R_1)^* \) then return the CFG for the star of \( grammar(R_1) \) as specified in 2.16

End

Then \( L(G) = A \) since at each step of the procedure a grammar is produced that accepts the same language as \( R \). The procedure is guaranteed to terminate since each regular expression \( R \) is finite and on each recursive call a smaller regular expression is passed to the procedure (since in the final three cases \( R_1 \) and \( R_2 \) are proper subcomponents of \( R \)).
2.35: Grammars in CNF whose languages have infinite length

Solution:

Let $G = (V, \Sigma, R, S)$ be a CFG in Chomsky normal form that contains $b$ variables and assume that $G$ generates some string with a derivation having at least $2^b$ steps. We want to show that $L(G)$ is infinite, or that there must exist a path along its parse tree that has a variable repeated. If there is a path that contains $b + 1$ variables, then by the pigeon-hole principle, one of the variables must be repeated. Thus, we want to show that if $G$ generates some string with a derivation having at least $2^b$ steps then it must have a parse tree with a height of $b + 1$.

For a given height, $h$, of a parse tree for a grammar in Chomsky normal form, that parse tree could have at most $2^h - 1$ steps in a derivation of a string. This is shown by induction on the height of the tree. The base case is $h = 1$ in which case the derivation is $S \rightarrow a$ for some $a \in \Sigma \cup \{\epsilon\}$. Thus, there is 1 step to the derivation and $1 = 2^1 - 1$. Assume that a parse tree of height $h$ has at most $2^h - 1$ steps to any derivation of a string. Then for a tree of height $h + 1$ it could have at most $2^h$ steps on the last level, corresponding to if all the elements on the previous level were variables. Then the maximum number of steps in the derivation are the maximum number of steps in the derivation on the last level plus the maximum number of steps for a derivation of the previous level which is given by the induction hypothesis. Thus, we get $2^h + 2^h - 1 = 2(2^h) - 1 = 2^{h+1} - 1$, which completes the induction.

Since $G$ has a string with a derivation of $2^b$ steps, the parse tree for that string must have a height of at least $b + 1$ (since with height $b$ it could have a derivation of at most $2^b - 1$ steps as shown above). Since the height of the tree is at least $b + 1$, then the longest path on that tree must have at least $b + 2$ nodes with only the last node a terminal. Thus, there must be at least $b + 1$ variables on that path. Let $A$ be a variable that repeats on that longest path, then the derivation of that string looks like $S \Rightarrow^* uAz' \Rightarrow^* uvAy'z' \Rightarrow^* uvxyz$ where $u, v, x, y, z \in \Sigma^*$ and $y', z' \in V^*$. By applying the derivation $A \Rightarrow^* vAy'$ $i$ times on that path for $A$ followed by the derivation $A \Rightarrow^* x$, we generate valid parse trees for the strings $uv^ixy^iz$ for $i \geq 0$. Thus, for each $i \geq 0$, $uv^ixy^iz \in L(G)$. Whence, there are an infinite number of strings in $L(G)$. 


(5) 1.54: \( F = \{ a^i b^j c^k | i, j, k \geq 0 \text{ and if } i = 1 \text{ then } j = k \} \), a nonregular language that works in the pumping lemma

Solution:

a. Show \( F \) is not regular

To see a solution that uses the reverse to show that \( F \) is not regular, see the solution given on the first midterm for 1c. We’ll give another solution here. For a contradiction assume \( F \) is regular, the \( F \cap \{ ab^i c^j | i, j \geq 0 \} = \{ ab^i c^j | i \geq 0 \} = G \) is regular since \( \{ ab^i c^j | i, j \geq 0 \} \) is the language of the regular expression \( ab^*c^* \), it is regular, and regular languages are closed under intersection. We will now use the pumping lemma to show that \( G \) cannot be regular, which will contradict the fact the we assumed \( F \) is regular. Let \( p \) be the pumping length of \( G \) and take \( s = ab^p c^p \in G \) with \( |s| > p \). Then there exists \( x, y, z \) such that \( s = xyz \) and  

\[
\begin{align*}
(1) & \quad x y^i z \in G \text{ for all } i \geq 0, \\
(2) & \quad |y| > 0 \text{ and (3) } |x y| \leq p. \quad \text{We will show that for any valid choice of } x y z \text{ that } x y^2 z \notin G.
\end{align*}
\]

Case 1: \( y = a \). Then \( x y^2 z = a^2 b^p c^p \notin G \).

Case 2: \( y = a b^k \) for some \( k < p - 1 \) from (3). Then, \( x y^2 z = a b^k a b^p c^p \notin G \).

Case 3: \( y = b^k \) for some \( k < p - 1 \) from (2) and (3). Then \( x y^2 z = a b^{p+k} c^p \notin G \).

Thus, for every possibility of \( x, y, z \), \( x y^2 z \notin G \). Therefore, \( G \) cannot be regular, a contradiction. Thus, we conclude that \( F \) cannot be regular.

b. Show that \( F \) acts like a regular language in the pumping lemma.

First I must give a \( p \). Take \( p = 2 \) although any \( p \) higher will work (but \( p = 1 \) will not, \( a a b c^2 \in F \) but pumping down \( a b c^2 \notin F \)). Now we must show for any string \( s \in F \), with \( |s| \geq 2 \), can be written as \( x y z \) such that (1) \( x y^i z \in G \) for all \( i \geq 0 \), (2) \( |y| > 0 \) and (3) \( |x y| \leq p \). Again, we will proceed by cases on the number of as in \( s \).

Case 1: \( s = b^i c^k \in F \) for some \( i, k \) such that \( j + k \geq 2 \) (so \( |s| \geq 2 \)). Then take \( x = \epsilon, y \) equal to the first symbol in \( s \), and \( z \) equal to the remaining symbols in \( s \). Then (2) and (3) hold and \( x y^i z = b^{j+i-1} c^k \in F \) if \( j \neq 0 \), or \( x y^i z = c^{k+j-1} \in F \) if \( j = 0 \), so (1) also holds.

Case 2: \( s = a b^i c^j \in F \) for some \( j \geq 1 \) (so \( |s| \geq 2 \)). Then take \( x = \epsilon, y = a \), and \( z = b^i c^j \). Then (2) and (3) hold and \( x y^i z = a^i b^j c^j \in F \), so (1) also holds.

Case 3: \( s = a^2 b^i c^k \in F \) for some \( j, k \geq 0 \) (so \( |s| \geq 2 \)). Then take \( x = \epsilon, y = a^2 \) (a won’t work here, because of pumping down), and \( z = b^i c^k \). Then (2) and (3) hold and \( x y^i z = a^{i+2b^j} c^k \in F \), so (1) also holds.

Case 4: \( s = a^i b^k c^m \in F \) for some \( j, k, m \geq 0 \) (so \( |s| \geq 2 \)). Then take \( x = \epsilon, y = a \), and \( z = a^{i-j} b^k c^m \). Then (2) and (3) hold and \( x y^i z = a^{i+2b^j} c^m \in F \), so (1) also holds.

Thus, in every case we see that the conditions of the pumping lemma hold. Therefore \( F \) satisfies the pumping lemma.

c. Explain why parts (a) and (b) do not contradict the pumping lemma.

The pumping lemma states that “If a language is regular then in can be pumped.” However, the converse of the statement, “If a language can be pumped then it is regular,” need not be true as shown in parts (a) and (b). This is because the converse of an implication is not equivalent to the implication (as can be shown with truth tables). Thus, the pumping lemma is not contradicted.