1. Give a context-free grammar (CFG) for each of the following languages. Any grammar is acceptable—including ambiguous grammars—as long as it has the correct language. The start symbol should be $S$.

(a) The set of all strings over the alphabet $\{1, 2, *, -\}$ representing valid products of (signed) integers where the expression evaluates to some negative odd value.

Example Strings in the Language:

- $-1$  
- $-1*121$  
- $221*-1*1121$

Strings not in the Language:

- $\varepsilon$  
- $-11*-121$  
- $12**22*112$

Solution:

$$S \rightarrow S \cdot P \mid P \cdot S \mid -D$$  
$$P \rightarrow S \cdot S \mid P \cdot P$$  
$$D \rightarrow 2D \mid 1D \mid 1$$

(b) The set of all strings over the alphabet $\{a, b, [, ], ,\}$ representing nested lists of $a$ and $b$’s. Nested lists are defined as comma separated sequences of elements enclosed with a pair of square brackets $[]$, where an element may be an $a$, $b$, or another list. Example Strings in the Language:

- $[]$  
- $[a, [[b]]]$  
- $[[a, b], [], a, b, [[a, [b]]]]$

Strings not in the Language:

- $\varepsilon$  
- $b$  
- $[a, b]$  
- $[a]$  
- $[aa]$

Solution:

$$S \rightarrow [L_1]$$  
$$L_1 \rightarrow L_2 \mid \varepsilon$$  
$$L_2 \rightarrow E \mid E, L_2$$  
$$E \rightarrow a \mid b \mid S$$
(c) The set of all strings over the alphabet \{0, 1\} where the number of 1's is exactly two times the number of 0's.

Example Strings in the Language:

\[
\begin{array}{ccc}
\varepsilon & 101 & 001111 \\
\end{array}
\]

Strings not in the Language:

\[
\begin{array}{ccc}
1 & 001 & 01100101 \\
\end{array}
\]

Solution:

\[
S \rightarrow Z11 \mid 1Z1 \mid 11Z \mid \varepsilon
\]

\[
Z \rightarrow 0S \mid S0 \mid 0
\]
2. Consider the following grammar for binary strings that involves the alphabet \{a, b\}:

\[
E \rightarrow Ea \mid Eb \mid aE \mid bE \mid T \\
T \rightarrow a \mid b \mid \varepsilon
\]

Is this grammar ambiguous or not? If yes, give an example of an expression with two different parse trees, draw the parse trees, and make the grammar unambiguous. If not, explain why it is unambiguous.

**Solution:**

Yes, the grammar is ambiguous. Consider the expression \(aa\) which has the following two parse trees:

One equivalent unambiguous grammar would be

\[
E \rightarrow Ea \mid Eb \mid \varepsilon
\]
3. (a) Eliminate left recursion from the following grammar:

\[
S \rightarrow S \cap S \mid (S) \mid T \\
T \rightarrow T \cup T \mid B \\
B \rightarrow \text{true} \mid \text{false} \mid B? \mid a[S]
\]

**Solution:**

\[
S \rightarrow TS' \mid (S)S' \\
T \rightarrow BT' \\
B \rightarrow \text{true}B' \mid \text{false}B' \mid a[S]B' \\
S' \rightarrow \cap SS' \mid \epsilon \\
T' \rightarrow \cup TT' \mid \epsilon \\
B' \rightarrow B' \mid \epsilon
\]

(b) Left factor the following grammar:

\[
S \rightarrow (E + x) \mid (E) \mid (E + E) \\
E \rightarrow (U - x) \mid U \mid (U - E) \\
U \rightarrow U0U \mid U1 \mid \epsilon
\]

**Solution:**

\[
S \rightarrow (ES' \\
S' \rightarrow +S'' \mid ) \\
S'' \rightarrow x \mid E) \\
E \rightarrow (U - E' \mid U \\
E' \rightarrow x \mid E) \\
U \rightarrow UU' \mid \epsilon \\
U' \rightarrow 1 \mid 0U
\]
4. Consider the following CFG, where the set of terminals is \{a, b, c, (, )\}:

\[
S \rightarrow U(T \mid b(U \\
T \rightarrow aSc \mid bU \mid ) \\
U \rightarrow cTb \mid aS) : \mid \varepsilon
\]

(a) Construct the FIRST sets for each of the nonterminals.

**Solution:**
- \( S \): \{a, b, c, (\}
- \( T \): \{a, b, )\}
- \( U \): \{a, c, \varepsilon\}

(b) Construct the FOLLOW sets for each of the nonterminals.

**Solution:**
- \( S \): \{c, ), $\}
- \( T \): \{b, c, ), $\}
- \( U \): \{b, c, (, ), $\}

(c) Construct the LL(1) parsing table for the grammar. Where applicable, list all possible productions for every parse table cell.

**Solution:**

<table>
<thead>
<tr>
<th>Nonterminal</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>(</th>
<th>)</th>
<th>:</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>U(T</td>
<td>b(U</td>
<td>U(T</td>
<td>U(T</td>
<td></td>
<td></td>
<td>$</td>
</tr>
<tr>
<td>( T )</td>
<td>aSc</td>
<td>bU</td>
<td></td>
<td></td>
<td></td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>( U )</td>
<td>aS</td>
<td>\varepsilon cTb, \varepsilon</td>
<td>\varepsilon</td>
<td>\varepsilon</td>
<td>\varepsilon</td>
<td>\varepsilon</td>
<td></td>
</tr>
</tbody>
</table>

(d) Is this grammar LL(1)? Explain. Hint: use your previous answers from parts c).

**Solution:** No, the grammar is not LL(1) since there is a production rule conflict for \( U \) when the lookahead terminal is "c". We can see this FIRST/FOLLOW conflict in part (c) with multiple productions in the \( <U, c> \) cell.

(e) Show the sequence of stack, input, and action configurations that occur during an LL(1) parse of the string "cbb() :". At the beginning of the parse, the stack should contain a single \( S \). Assume conflicts will lead to errors. The acceptable actions include: "out <production>", "match <terminal>", "accept", and "error".

**Solution:**

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S $)</td>
<td>cbb():$</td>
<td>out ( S \rightarrow U(T) )</td>
</tr>
<tr>
<td>( U(T $)</td>
<td>cbb():$</td>
<td>error</td>
</tr>
</tbody>
</table>

(f) Now assume we have the precedence that productions with more right-hand-side symbols will take priority (i.e. \( T \rightarrow aSc \triangleright \triangleright T \rightarrow bU \) since \( aSc \) has 3 right-hand-side symbols and \( bU \) only has 2 right-hand-side symbols). Show the updated stack, input, and action configurations that occur during a parse of the string "cbb() :".

**Solution:**
<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$cbb():$</td>
<td>out $S \rightarrow U(T$</td>
</tr>
<tr>
<td>$U(T$</td>
<td>$cbb():$</td>
<td>out $U \rightarrow cTb$</td>
</tr>
<tr>
<td>$cTb(T$</td>
<td>$cbb():$</td>
<td>match $c$</td>
</tr>
<tr>
<td>$Tb(T$</td>
<td>$bb():$</td>
<td>out $T \rightarrow bU$</td>
</tr>
<tr>
<td>$bUb(T$</td>
<td>$bb():$</td>
<td>match $b$</td>
</tr>
<tr>
<td>$Ub(T$</td>
<td>$b():$</td>
<td>out $U \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>$b(T$</td>
<td>$b():$</td>
<td>match $b$</td>
</tr>
<tr>
<td>$(T$</td>
<td>$(():$</td>
<td>match $($</td>
</tr>
<tr>
<td>$T$</td>
<td>$):$</td>
<td>out $T \rightarrow )$</td>
</tr>
<tr>
<td>$:$</td>
<td>$):$</td>
<td>match $)$</td>
</tr>
<tr>
<td>$:$</td>
<td>$:$</td>
<td>match $:$</td>
</tr>
<tr>
<td>$:$</td>
<td>$:$</td>
<td>accept</td>
</tr>
</tbody>
</table>
5. Consider the following grammar $G$ over the alphabet $\{x, =\}$:

\[
\begin{align*}
S' & \rightarrow S \\
S & \rightarrow L = R \\
S & \rightarrow R \\
L & \rightarrow x \\
R & \rightarrow L
\end{align*}
\]

You want to implement $G$ using an SLR(1) parser. Note that we have already added the $S' \rightarrow S$ production for you.

(a) Construct the DFA of the LR(0) machine, and identify all conflicting states and conflicts that prevent the grammar from being LR(0).

**Solution:**

There is one conflicting state that has a shift-reduce conflict:

\[
S \rightarrow L = R
\]

(b) Now, for each conflicting state in the DFA that prevents it from being LR(0), identify the FOLLOW sets of the left-hand nonterminals. Is the grammar SLR(1)? Explain. Your explanation must reference at least one of the identified FOLLOW sets from each conflicting DFA state.

**Solution:**

For the conflicting state we have $\text{FOLLOW}(S) = \{$ and $\text{FOLLOW}(R) = \{$. The grammar is SLR(1) since there is no longer a conflict. The conflict gets resolved since $\text{FOLLOW}(R) = \{$ so if the next look-ahead terminal is a $\$ the parser should reduce, and if the look-ahead terminal is a $=$ the parser will shift.