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 $\{3, 4, *, -\}$ representing valid products of integers where the expression evaluates to some positive odd value.

Example Strings in the Language:

$$3 \quad 3*33 \quad 3*3*-343$$

Strings not in the Language:

$$\varepsilon \quad -3 \quad -3*3*-3344$$

Solution:

$$S \rightarrow S \cdot S \mid P \mid N \cdot N \mid N \cdot S \cdot N$$
$$P \rightarrow 4P \mid 3P \mid 3$$
$$N \rightarrow -P$$

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

$$\lbrack\rbrack \quad \lbrack a, [b,a], a\rbrack \quad \lbrack[a,b],[a,a],[\lbrack,b,b]\rbrack$$

Strings not in the Language:

$$\varepsilon \quad a \quad \lbrack a, b, \rbrack \quad [a] \quad [a, [b, a]]$$

Solution:

$$S \rightarrow [T]$$
$$T \rightarrow \varepsilon \mid U$$
$$U \rightarrow E, E \mid E, E, U$$
$$E \rightarrow S \mid a \mid b$$
(c) The set of all strings over the alphabet \{0, 1\} where the number of 1's is more than the number of 0's.

Example Strings in the Language:

- 1
- 101
- 001111

Strings not in the Language:

- \epsilon
- 0
- 01100101

Solution:

\[
S \rightarrow T1T \\
T \rightarrow T1T0T \mid T1T \mid T0T1T \mid \epsilon
\]

(d) The set of all strings over the alphabet \{0, 1\} in the language \( L : \{1^i0^j1^k \mid j \geq i + k\} \).

Example Strings in the Language:

- \epsilon
- 0
- 100011

Strings not in the Language:

- 1
- 010
- 101

Solution:

\[
S \rightarrow TZU \\
T \rightarrow 1T0 \mid \epsilon \\
U \rightarrow 0U1 \mid \epsilon \\
Z \rightarrow Z0 \mid \epsilon
\]
2. Consider the following grammar for if-then-else expressions that involve a variable $x$:

$$E \rightarrow \text{if } x \text{ then } E \mid M$$
$$M \rightarrow \text{if } x \text{ then } M \text{ else } E \mid x$$

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

**Solution:** The grammar is ambiguous. To see why, consider the expression “if $x$ then if $x$ then $x$ else if $x$ then $x$ else $x$”. This expression has two parse trees under this grammar, shown below:
3. (a) Left factor the following grammar:

\[
S \rightarrow T \mid T + T \mid T * T
\]
\[
T \rightarrow Ta \mid Tb \mid eU
\]
\[
U \rightarrow U0 \mid U1 \mid \varepsilon
\]

Solution:

\[
S \rightarrow TS'
\]
\[
S' \rightarrow \varepsilon \mid +T \mid *T
\]
\[
T \rightarrow TT' \mid cU
\]
\[
T' \rightarrow a \mid b
\]
\[
U \rightarrow UU' \mid \varepsilon
\]
\[
U' \rightarrow 0 \mid 1
\]

(b) Eliminate left recursion from the following grammar:

\[
S \rightarrow S + S \mid (S) \mid T
\]
\[
T \rightarrow UUb \mid Ta
\]
\[
U \rightarrow TTc \mid c
\]

Solution:

\[
S \rightarrow (S)S' \mid TS'
\]
\[
S' \rightarrow +SS' \mid \varepsilon
\]
\[
T \rightarrow cUbT'
\]
\[
T' \rightarrow TeUbT' \mid aT' \mid \varepsilon
\]
\[
U \rightarrow TTc \mid c
\]
4. Consider the following CFG, where the set of terminals is \( \{a, b, c, <, >\} \):

\[
S \rightarrow T < U \mid b > U \\
T \rightarrow aS < S \mid cU > b \\
U \rightarrow T a < Sb
\]

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

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

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

**Solution:**
- \( S : \{a, b, <, \$_\} \)
- \( T : \{a, <\} \)
- \( U : \{a, b, <, \$_\} \)

(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></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>&lt;</th>
<th>&gt;</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>( T &lt; U )</td>
<td>( b &gt; U )</td>
<td>( T &lt; U )</td>
<td>( T &lt; U )</td>
<td>( $_)</td>
<td></td>
</tr>
<tr>
<td>( T )</td>
<td>( aS &lt; S )</td>
<td>( cU )</td>
<td>( $_)</td>
<td>( &gt; b )</td>
<td>( $_)</td>
<td></td>
</tr>
<tr>
<td>( U )</td>
<td>( $_)</td>
<td>( $_)</td>
<td>( Sb )</td>
<td>( Ta )</td>
<td>( $_)</td>
<td></td>
</tr>
</tbody>
</table>

(d) Show the sequence of stack, input, and action configurations that occur during an LL(1) parse of the string “\( > b << ba \)”. At the beginning of the parse, the stack should contain a single \( S \). 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>$$_</td>
<td>( b &lt;&lt; ba )</td>
<td>output ( S \rightarrow T &lt; U )</td>
</tr>
<tr>
<td>( T &lt; U $</td>
<td>( b &lt;&lt; ba )</td>
<td>output ( T \rightarrow b )</td>
</tr>
<tr>
<td>( b &lt; U $_</td>
<td>( b &lt;&lt; ba )</td>
<td>match &gt;</td>
</tr>
<tr>
<td>( &lt; U $_</td>
<td>( b &lt;&lt; ba )</td>
<td>match &lt;</td>
</tr>
<tr>
<td>( U$_</td>
<td>( b &lt;&lt; ba )</td>
<td>output ( U \rightarrow Ta )</td>
</tr>
<tr>
<td>( Ta$_</td>
<td>( b &lt;&lt; ba )</td>
<td>match &gt;</td>
</tr>
<tr>
<td>( b$_</td>
<td>( b &lt;&lt; ba )</td>
<td>match b</td>
</tr>
<tr>
<td>( a$_</td>
<td>( a$_</td>
<td>match a</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>accept</td>
</tr>
</tbody>
</table>
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 first state of the LR(0) machine, compute the FOLLOW sets of \( A \) and \( B \), and point out the conflicts that prevent the grammar from being SLR(1).

**Solution:** Here is the first state of the LR(0) machine:

\[
\begin{align*}
S' & \rightarrow S \\
S & \rightarrow Aa \\
S & \rightarrow Bb \\
A & \rightarrow Ac \\
A & \rightarrow \varepsilon \\
B & \rightarrow Bc \\
B & \rightarrow \varepsilon 
\end{align*}
\]

We have that \( \text{FOLLOW}(A) = \{a, c\} \) and \( \text{FOLLOW}(B) = \{b, c\} \). We have a reduce-reduce conflict between production 5 (\( A \rightarrow \varepsilon \)) and production 7 (\( B \rightarrow \varepsilon \)), so the grammar is not SLR(1).

(b) Show modifications to production 4 (\( A \rightarrow Ac \)) and production 6 (\( B \rightarrow Bc \)) that make the grammar SLR(1) while having the same language as the original grammar \( G \). Explain the intuition behind this result.

**Solution:** We change productions 4 and 6 to be right recursive, as follows:

\[
\begin{align*}
A & \rightarrow cA \\
B & \rightarrow cB 
\end{align*}
\]

As a result, \( c \) is no longer in the follow set of either \( A \) nor \( B \). Since the follow sets are now disjoint, the reduce-reduce conflict in the SLR(1) table is removed. Intuitively, using right recursion causes the parser to defer the decision of whether to reduce a string of \( c \)'s to \( A \)'s or \( B \)'s. When we reach the end of the input, the final character gives us enough information to determine which reduction to perform.