1. An ambiguous grammar for expressions is
\[
E' \rightarrow E -
\]
\[
E \rightarrow E + E
\]
\[
E \rightarrow E * E
\]
\[
E \rightarrow ID
\]

(a) Use Earley’s algorithm to parse “ID + ID * ID” for the grammar by filling in the chart below.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>E' \rightarrow \bullet \dash 0</td>
<td>E \rightarrow ID\bullet 0</td>
<td>E \rightarrow E + \bullet E, 0</td>
<td>E \rightarrow ID\bullet 2</td>
<td>E \rightarrow E * \bullet E, 2</td>
<td>E \rightarrow ID\bullet 4</td>
</tr>
<tr>
<td>b</td>
<td>E \rightarrow \bullet E + E, 0</td>
<td>E' \rightarrow E\bullet \dash, 0</td>
<td>E \rightarrow \bullet E + E, 0</td>
<td>E \rightarrow E + E\bullet 0</td>
<td>E \rightarrow E * E\bullet 0</td>
<td>E \rightarrow E * E\bullet 2</td>
</tr>
<tr>
<td>c</td>
<td>E \rightarrow \bullet E * E, 0</td>
<td>E \rightarrow E \bullet + E, 0</td>
<td>E \rightarrow \bullet E * E, 2</td>
<td>E \rightarrow E \bullet * E, 2</td>
<td>E \rightarrow E \bullet * E, 4</td>
<td>E \rightarrow E * E\bullet 0</td>
</tr>
<tr>
<td>d</td>
<td>E \rightarrow \bullet ID, 0</td>
<td>E' \rightarrow E\bullet * E\dash, 0</td>
<td>E \rightarrow \bullet ID, 2</td>
<td>E \rightarrow E \bullet * E, 2</td>
<td>E \rightarrow E \bullet * E, 0</td>
<td>E \rightarrow E \bullet * E, 0</td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Draw the two accepting parse trees and identify that there are ambiguities in the grammar. Explain what are the ambiguities and how to use precedence rules to solve them.

There are two ways to reach \( E' \rightarrow E\bullet \dash, 0 \):

\[
(a_0, b_0, d_0, d_1, c_1, a_2, c_2, d_2, a_3, b_3, g_3, b_4, e_4, a_5, c_5, i_5) \quad (ID + ID) * ID
\]

and

\[
(a_0, b_0, d_0, d_1, c_1, a_2, c_2, d_2, a_3, d_3, a_4, e_4, a_5, b_5, f_5, i_5) \quad ID + (ID * ID)
\]

Each will produce a different parse tree.
2. Unambiguous grammar rewriting

(a) Rewrite the grammar in problem.1 so that it’s unambiguous and has the sensible precedence and associativity.

\[
E \rightarrow E + T \mid T \\
T \rightarrow T * F \mid F \\
F \rightarrow \text{ID}
\]

(b) Use shift-reduce parsing for the string “\text{ID + ID * ID}”. You should get a unique parse tree this time.

\[
\epsilon / \text{ID} + \text{ID} * \text{ID} \Rightarrow_s \text{ID} / + \text{ID} * \text{ID} \Rightarrow_r \text{F} / + \text{ID} * \text{ID} \\
\Rightarrow_r \text{E} / + \text{ID} * \text{ID} \Rightarrow_s \text{E} + / \text{ID} * \text{ID} \Rightarrow_s \text{E} + \text{ID} / * \text{ID} \Rightarrow_r \text{E} + \text{F} / * \text{ID} \Rightarrow_r \\
\text{E} + \text{T} / * \text{ID} \Rightarrow_s \text{E} + \text{T} * / \text{ID} \Rightarrow_s \text{E} + \text{T} * \text{ID} / \epsilon \Rightarrow_r \text{E} + \text{T} * \text{F} / \epsilon \Rightarrow_r \text{E} + \text{T} / \epsilon \Rightarrow_r \text{E} / \epsilon
\]

3. (Challenge question) Prove that during shift-reduce parsing, we can only reduce the topmost items in the stack (i.e. we don’t need to worry about reducing something in the middle, hence the usage of a stack). Hint: shift-reduce parsing is finding the rightmost derivation in reverse.

(Proof by contradiction)
Suppose the stack is \(\alpha \beta \gamma\), where the greek letters are strings of 0 or more symbols and capital roman letters are non-terminal symbols. Suppose we want to reduce \(\beta\) using the rule \(B \rightarrow \beta\) when it is not at the top of the stack (i.e. \(\gamma \neq \epsilon\)). Then \(\gamma\) can be either 1) a string of terminals, or 2) \(\gamma = \gamma' A \gamma''\). For case 1, we could have reduced \(\beta\) before shifting on the terminal symbols, since they don’t help in any way of matching the right hand side of a rule. Case 2 is problematic, because shift-reduce parsing finds the rightmost derivation in reverse order, but

\[(\text{input}) \Leftarrow^* \alpha \beta \gamma' A \gamma'' \Leftarrow \alpha B \gamma' A \gamma'' \Leftarrow^* S\]

is not a rightmost derivation. Thus, a reduction in the middle of the stack cannot occur.