1 Bottom-up Parsing Solutions

1. We can see that there would be three shift-reduce conflicts in our machine (states 1, 3, 5) if weren’t for the attached follow sets. Thus this grammar is not LR(0).

2. Algorithm:
   - We begin with 0 on the state stack.
   - We use the top of the state stack and the first letter of input to determine the next action.
   - If it is a shift, move the top of the input to the symbol stack and push the new state to the state stack.
   - If it is a reduce, remove a number of items equal to the length of the right hand side of the rule we reduce by, then shift the nonterminal.
   - If the entry does not exist, reject.
   - If we shift ⊣, accept. If we want, we can do a final reduction for $S'$, but this is unnecessary, hence why the machine has no actions for state 6.

3. The reversed rightmost derivation can be obtained by reading the rules used in reductions from top to bottom.
bottom, and then adding the $S'$ rule at the end. Here is the reversed derivation in CFG-R format:

\[
\begin{align*}
X \\
X \ p \ X \\
Y \ q \\
S \ X \ Y \\
S \ S \ a \ b \\
S' \ ⊢ \ S \ ⊣
\end{align*}
\]

Furthermore, we can obtain a rightmost derivation by reading the symbol stack concatenated with the remaining input from bottom to top:

\[
\begin{align*}
S' &⇒ S \ ⊣ \\
⇒ S &⊢ S_{ab} \ ⊣ \\
⇒ X &⊢ XY_{ab} \ ⊣ \\
⇒ X &⊢ Xq_{ab} \ ⊣ \\
⇒ &⊢ pXq_{ab} \ ⊣ \\
⇒ &⊢ pq_{ab} \ ⊣
\end{align*}
\]

We can obtain a parse tree from this derivation by looking at what rule was used to expand each non-terminal symbol in the derivation.

![Parse Tree](image-url)
2 Top-down Parsing Solutions

Consider the following context-free grammar $G$:

\[
\begin{align*}
S' & \rightarrow \vdash S \vdash & (0) \\
S & \rightarrow aXYb & (1) \\
S & \rightarrow XY & (2) \\
X & \rightarrow pX & (3) \\
X & \rightarrow \epsilon & (4) \\
Y & \rightarrow q & (5) \\
Y & \rightarrow \epsilon & (6)
\end{align*}
\]

1. Algorithm:

- If there is a terminal on top of the stack and the same terminal is in the input, we consume it.
- If there is a nonterminal on top of the stack and the predictor table has an entry at the position of the nonterminal and the letter on top of the stack, we expand the nonterminal using that rule.
- We accept upon reading EOF.
- In any other case, we reject.

<table>
<thead>
<tr>
<th>Action</th>
<th>Consumed Input</th>
<th>Stack</th>
<th>Remaining Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
<td></td>
<td>$S'$ $\vdash$ appqb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>expand $S' \rightarrow S \vdash$</td>
<td>$\vdash S \vdash$</td>
<td>$\vdash S \vdash$ appqb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>match $\vdash$</td>
<td>$\vdash$</td>
<td>$S \vdash$ appqb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>expand $S \rightarrow aXYb$</td>
<td>$\vdash$ aXYb $\vdash$</td>
<td>appqb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>match $a$</td>
<td>$\vdash a$</td>
<td>$XYb \vdash$ appqb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>expand $X \rightarrow pX$</td>
<td>$\vdash a$</td>
<td>$pXYb \vdash$ appqb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>match $p$</td>
<td>$\vdash ap$</td>
<td>$XYb \vdash$ pqb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>expand $X \rightarrow pX$</td>
<td>$\vdash ap$</td>
<td>$pXYb \vdash$ pqb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>match $p$</td>
<td>$\vdash app$</td>
<td>$XYb \vdash$ qb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>expand $X \rightarrow aXYb$</td>
<td>$\vdash app$</td>
<td>$Yb \vdash$ qb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>expand $Y \rightarrow q$</td>
<td>$\vdash appq$</td>
<td>$qb \vdash$ qb $\vdash$</td>
<td></td>
</tr>
<tr>
<td>match $q$</td>
<td>$\vdash appq$</td>
<td>$b \vdash$ b $\vdash$</td>
<td></td>
</tr>
<tr>
<td>match $b$</td>
<td>$\vdash appqb$</td>
<td>$\vdash$ \vdash</td>
<td></td>
</tr>
</tbody>
</table>

Reading the production rules that were applied from top to bottom gives a leftmost derivation. We can use this derivation to obtain the parse tree.
2. If we modify the predictor table accordingly for the row for $S$, notice that two of the cells have two entries each. Since we require that each cell in the predictor table has at most one entry, this grammar is not LL(1). In this case, the issue arises from the fact that left-recursive rules (i.e. $S \rightarrow Sab$) are not LL(1).