1 Top-Down Parsing

1.1 Definitions

- LL(1) stands for *Left-to-right* scan of input, *Left-canonical* derivations produced, 1 symbol of lookahead.
- An LL(1) parser is a *top-down* parser; it begins from the start symbol and finds a derivation for the input string.
- A grammar that can be parsed using the LL(1) algorithm is called an LL(1) grammar.
- The LL(1) prediction function is defined as:

\[
\text{Predict}(A, a) = \{ A \rightarrow \gamma | a \in \text{First}(\gamma) \lor (\text{Nullable}(\gamma) \land a \in \text{Follow}(A)) \}
\]

\[
\text{First}(\gamma) = \{ b | \gamma \Rightarrow b\beta \text{ for some } \beta \}
\]

\[
\text{Follow}(A) = \{ c | S' \Rightarrow \alpha Ac\beta \text{ for some } \alpha, \beta \}
\]

\[
\text{Nullable}(\gamma) = \gamma \Rightarrow \epsilon
\]

- A grammar is LL(1) iff $|\text{Predict}(A, a)| \leq 1 \forall A, a$.

1.2 The LL(1) Algorithm

Consider the following context-free grammar $G_1$ and associated predictor table:

1.2.1 $G_1$

\[
\begin{align*}
S' & \rightarrow \vdash S \vdash \quad (0) \\
S & \rightarrow aXYb \quad (1) \\
S & \rightarrow XY \quad (2) \\
X & \rightarrow pX \quad (3) \\
X & \rightarrow \epsilon \quad (4) \\
Y & \rightarrow q \quad (5) \\
Y & \rightarrow \epsilon \quad (6)
\end{align*}
\]
1.2.2 \( G_1 \) LL(1) Predictor Table

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>S'</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>X</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Y</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

1.2.3 Problem

Using the LL(1) predictor table for \( G_1 \), perform a top-down parse of the string \( a\overset{\vdash}{\rightarrow}pqb \overset{\vdash}{\rightarrow} \) and draw the parse tree.

2 Bottom-up Parsing

2.1 Definitions

- LR(1) stands for Left-to-right scan of input, Right-canonical derivations produced, 1 symbol of lookahead. The S in SLR(1) stands for Simplified.
- An LR(1) parser is a bottom-up parser; it begins with the input string and replaces right hand sides of rules with left hand sides until the start symbol is produced.
- A grammar that can be parsed using the LR(1) algorithm is called an LR(1) grammar.
- An item is a production with a dot (\( \bullet \)) somewhere on the right-hand side, and represents a partially-completed rule.
- A shift-reduce conflict occurs when a state in the LR machine has both completed and partially completed rules with overlapping follow sets.
- A reduce-reduce conflict occurs when a state in the LR machine has two different completed rules with overlapping follow sets.

2.2 The SLR(1) Algorithm

```
push \( q_0 \)
for each \( a \) in \( \vdash \) input \( \vdash \)
  while (Reduce[stack.top, \( a \)] = \( A \rightarrow \gamma \))
    pop \( 2*|\gamma| \) times
    state \( \leftarrow \) stack.top
    push \( A \)
    push Trans[state, \( A \)]
    state \( \leftarrow \) stack.top
  if (Trans[state, \( a \)] = ERROR) reject
  push \( a \)
  push Trans[state, \( a \)]
accept
```
2.3 Bottom-up Parsing Using SLR(1)

Consider the following context-free grammar \( G_3 \) and its corresponding SLR(1) shift/reduce machine, given in both bubble diagram and table form:

2.3.1 \( G_3 \)

\[
\begin{align*}
S' & \rightarrow \vdash S \vdash \quad (0) \\
S & \rightarrow S a b \quad (1) \\
S & \rightarrow X Y \quad (2) \\
X & \rightarrow p X \quad (3) \\
X & \rightarrow \epsilon \quad (4) \\
Y & \rightarrow q \quad (5) \\
Y & \rightarrow \epsilon \quad (6)
\end{align*}
\]

2.3.2 \( G_3 \) SLR(1) Machine Table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\vdash</td>
<td>shift 1</td>
<td>a</td>
<td>reduce 2</td>
<td>5</td>
<td>a</td>
<td>reduce 4</td>
<td>8</td>
<td>a</td>
<td>shift 4</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>reduce 4</td>
<td>2</td>
<td>\vdash</td>
<td>reduce 2</td>
<td>5</td>
<td>p</td>
<td>shift 5</td>
<td>8</td>
<td>\vdash</td>
</tr>
<tr>
<td>1</td>
<td>p</td>
<td>shift 5</td>
<td>3</td>
<td>a</td>
<td>reduce 6</td>
<td>5</td>
<td>q</td>
<td>reduce 4</td>
<td>9</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>q</td>
<td>reduce 4</td>
<td>3</td>
<td>q</td>
<td>shift 7</td>
<td>5</td>
<td>\vdash</td>
<td>reduce 4</td>
<td>9</td>
<td>q</td>
</tr>
<tr>
<td>1</td>
<td>\vdash</td>
<td>reduce 4</td>
<td>3</td>
<td>\vdash</td>
<td>reduce 6</td>
<td>5</td>
<td>X</td>
<td>shift 9</td>
<td>9</td>
<td>\vdash</td>
</tr>
<tr>
<td>1</td>
<td>S</td>
<td>shift 8</td>
<td>3</td>
<td>Y</td>
<td>shift 2</td>
<td>7</td>
<td>a</td>
<td>reduce 5</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>shift 3</td>
<td>4</td>
<td>b</td>
<td>shift 10</td>
<td>7</td>
<td>\vdash</td>
<td>reduce 5</td>
<td>10</td>
<td>\vdash</td>
</tr>
</tbody>
</table>
2.3.3 \( G_3 \) SLR(1) Machine Bubble Diagram

2.3.4 Problem

Use the shift/reduce table and grammar given above to parse the string \( \overrightarrow{pqab} \). Write the reversed right-canonical derivation for the string as well as the parse tree.