1 Top-down Parsing

1.1 Definitions

- LL(1) stands for *Left-to-right* scan of input, *Leftmost* derivations, 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 for a grammar \( G = (N, \Sigma, P, S) \) is defined as:
  \[
  \text{Predict}(A, a) = \{ A \rightarrow \gamma | a \in \text{First}(\gamma) \lor (\text{Nullable}(\gamma) \land a \in \text{Follow}(A)) \}
  \]

  Where:

  - \( \text{First}(\gamma) = \{ b | \gamma \Rightarrow^* b\beta \text{ for some } \beta \} \)
  - \( \text{Follow}(A) = \{ c | S' \Rightarrow^* \alpha A c \beta \text{ for some } \alpha, \beta \} \)
  - \( \text{Nullable}(\gamma) = \gamma \Rightarrow^* \epsilon \)

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

1.2 Problems

Consider the following context-free grammar \( G \):

\[
S' \rightarrow \vdash S \vdash \\
S \rightarrow aXYb \\
S \rightarrow XY \\
X \rightarrow pX \\
X \rightarrow \epsilon \\
Y \rightarrow q \\
Y \rightarrow \epsilon
\]

1. Create the LL(1) predictor table for \( G \).
2. Perform a top-down parse of the string ⊢ appqb ⊣ and draw the parse tree.

3. Consider the grammar $G'$, $G$ with rule 2 ($S \rightarrow aXYb$) replaced by the rule $S \rightarrow Sab$. Why is it impossible to parse this grammar with a top-down LL(1) parser? (That is, what aspect of the grammar makes it not LL(1)?)

## 2 Bottom-up Parsing

### 2.1 Definitions

- A bottom-up parser; it begins with the input string and replaces the right hand side of rules with the corresponding nonterminal on the left hand side until the start symbol is produced. In other words, it finds a reversed derivation of the input, ending at the start symbol.
- LR(0) stands for Left-to-right scan of input, Rightmost derivations, no lookahead.
- An item is a production with a dot (●) 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 and we cannot determine whether to reduce or shift.
- A reduce-reduce conflict occurs when a state in the LR machine has two different completed rules and we cannot determine which rule to reduce by.
- A grammar is LR(0) if its corresponding machine has no conflicts
- An SLR(1) parser (short for Simplified LR(1)) is an LR(0) parser where every finished rule is also given the follow set of its left-hand side as lookaheads. This helps drastically reduce the number of shift-reduce or reduce-reduce conflicts, since there is now only a conflict if the two possibilities (either shifting and reducing or reducing with two different rules) both occur in the same state and also share a common lookahead symbol. A grammar is SLR(1) if it can be parsed by an SLR(1) parser.

There are other LR(1)-family algorithms, notably LR(1) and LALR(1), which we do not cover in this course. The only difference between these algorithms is how the machine is constructed: the algorithm that uses the machine is the same in every case.

### 2.2 Problems

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

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

```
2.2.1 SLR(1) Machine Bubble Diagram

```

Note: the state numbers have nothing to do with the rule numbers

2.2.2 SLR(1) Machine Table

For each set of three columns, the left represents the state number, the middle the next input symbol, the last the action to take. If the action is "shift x", x denotes the state to move to. If the action is "reduce y", y denotes the rule to reduce by.

<table>
<thead>
<tr>
<th>State</th>
<th>Shift/Reduce Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>⊢ shift 1</td>
</tr>
<tr>
<td>1</td>
<td>⊢ a reduce 4</td>
</tr>
<tr>
<td>2</td>
<td>⊢ ⊥ reduce 2</td>
</tr>
<tr>
<td>3</td>
<td>S → X • Y</td>
</tr>
<tr>
<td>4</td>
<td>S → Sa • b</td>
</tr>
<tr>
<td>5</td>
<td>X → p • X</td>
</tr>
<tr>
<td>6</td>
<td>S' → ⊥ S • ⊥</td>
</tr>
<tr>
<td>7</td>
<td>Y → q</td>
</tr>
<tr>
<td>8</td>
<td>S' → ⊥ S • ⊥</td>
</tr>
<tr>
<td>9</td>
<td>X → pX•</td>
</tr>
<tr>
<td>10</td>
<td>S → Sab•</td>
</tr>
</tbody>
</table>

1. Is the grammar LR(0)?
2. Use the shift/reduce table and grammar given above to parse the string ⊢ pqab ⊥.
3. Write the reversed rightmost derivation for the string, as well as the parse tree.