1 Top-down Parsing (LL(1))

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 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 Problems

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

\[
\begin{align*}
S' &\rightarrow \vdash S \vdash \quad (1) \\
S &\rightarrow aXYb \quad (2) \\
S &\rightarrow XY \quad (3) \\
X &\rightarrow pX \quad (4) \\
X &\rightarrow \epsilon \quad (5) \\
Y &\rightarrow q \quad (6) \\
Y &\rightarrow \epsilon \quad (7)
\end{align*}
\]

1. Create the LL(1) predictor table for \(G\).
2. Perform a top-down parse of the string \(\vdash appqb \vdash\) 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 (LR(0)/SLR(1)/LR(1))

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

- To eliminate many conflicts, we can attach the follow set to each reducible item in the LR(0) machine. This construction is called an SLR(1) parser, which allows for 1 symbol of lookahead, and can parse strictly more grammars than an LR(0) parser. The S in SLR(1) stands for *Simplified*.

- If we use the next input symbol to make a decision rather than the weaker condition that the next input symbol should be in the follow set, we now have a LR(1) parser. LR(1) parsers are strictly more powerful than SLR(1) parsers. A grammar that can be parsed using the LR(1) algorithm is called an LR(1) grammar.

  Note: The algorithm used for LR(1) and SLR(1) parsers is the same, the only difference is the underlying machine.

2.2 Problems

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

\[
\begin{align*}
S' & \rightarrow \vdash S \vdash \\
S & \rightarrow Sab \\
S & \rightarrow XY \\
X & \rightarrow pX \\
X & \rightarrow \epsilon \\
Y & \rightarrow q \\
Y & \rightarrow \epsilon
\end{align*}
\]
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></th>
<th></th>
<th>action</th>
<th></th>
<th>action</th>
<th></th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>shift 1</td>
<td>2</td>
<td>a</td>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>reduce 4</td>
<td>2</td>
<td>⊥</td>
<td>5</td>
<td>p</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>shift 5</td>
<td>3</td>
<td>a</td>
<td>5</td>
<td>q</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>reduce 4</td>
<td>3</td>
<td>q</td>
<td>5</td>
<td>⊥</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>⊥</td>
<td>3</td>
<td>shift 7</td>
<td>5</td>
<td>⊥</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>shift 8</td>
<td>3</td>
<td>Y</td>
<td>5</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>shift 3</td>
<td>4</td>
<td>b</td>
<td>7</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>shift 10</td>
<td>4</td>
<td>shift 10</td>
<td>7</td>
<td>⊥</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.