Building the Shift–Reduce Table

We show how to construct the table for bottom-up parsing provided the LR(1) grammar has the right form.
We now describe how to construct the table used in bottom-up parsing of LR(1) grammars.

This process has two main phases:

(1) determining the states and the shifts, and
(2) determining the reductions.
We use the arithmetic grammar as an example.

1: \[ E \rightarrow E + T \]
2: \[ E \rightarrow T \]
3: \[ T \rightarrow T \times F \]
4: \[ T \rightarrow F \]
5: \[ F \rightarrow (E) \]
6: \[ F \rightarrow n \]
The key construct is an **item**: a production with a certain point on RHS marked. We use a dot as the **marker** and enclose whole item in square brackets.

For example, item \([E \rightarrow E+ \cdot T]\) is production \(E \rightarrow E+T\) with a marker before the \(T\).

If production has RHS of length \(n\), then there are \(n + 1\) corresponding items.

An **initial** item has marker at the left end; a **completed** item has marker at the right end.
The intended meaning of an item is that the symbols to the left of the marker have already been read, or reduced to, and they are the top of the stack.

For our parser,

*each state is a set of items, called an item-set.*

(Note that if construction goes wrong at any point, that is a proof that the grammar is not in right form.)
To determine the item that comes after an item:

1. **Advance the marker.** So if item is $[E \to \cdot (E)]$ and next symbol is $($, we go to item $[E \to (\cdot E)]$.

2. **Take the closure.** For an item, the **closure** is set of items as follows: If marker precedes a variable, then closure adds all initial items starting with that variable. If any new item has marker preceding a variable, then add all initial items with that variable. And repeat. (If marker precedes a terminal, there is nothing to add.)
For the arithmetic grammar, consider the item 
\[ T \rightarrow T \cdot \times F \].

On the symbol \( \times \), the result is 
\[ T \rightarrow T \times \cdot F \].

The closure adds all initial items beginning with \( F \): 
that is, 
\[ F \rightarrow \cdot (E) \]  and \[ F \rightarrow \cdot n \].
Determining the next item-set. Let \( Q \) be an item-set and \( \sigma \) a symbol. The next item-set is constructed by:

For each item in \( Q \) where marker precedes \( \sigma \):
1. Add item that results from advancing marker past \( \sigma \).
2. Add closure of resultant item.
The Start Item-Set

We add new start variable to CFG with its only production taking it to the old start variable.

The start item-set is the initial item for the (new) start variable, and its closure.

For the example CFG: The start item is \([E' \rightarrow \cdot E]\). Its closure adds all initial items starting with \(E\), which adds all initial items starting with \(T\), which adds all initial items starting with \(F\):

\[
\{ [E' \rightarrow \cdot E], [E \rightarrow \cdot E+T], [E \rightarrow \cdot T], [T \rightarrow \cdot T \times F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot n] \}.
\]
We now determine the states and the shifts.

Start with the start item-set. Consider every possible next symbol: terminal and variable. For each symbol, determine the next item-set. Then for each item-set so created, do the same thing. Repeat until no new item-set is created.

Here is the calculation of item-sets for the arithmetic CFG:
<table>
<thead>
<tr>
<th>State</th>
<th>first occurs</th>
<th>Items found by moving marker</th>
<th>items added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>start</td>
<td>$[E' \rightarrow \cdot E]$</td>
<td>$E, T, F$</td>
</tr>
<tr>
<td>1</td>
<td>0, (</td>
<td>$[F \rightarrow (\cdot E)]$</td>
<td>$E, T, F$</td>
</tr>
<tr>
<td>2</td>
<td>0, n</td>
<td>$[F \rightarrow n\cdot]$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0, $E$</td>
<td>$[E' \rightarrow E\cdot], [E \rightarrow E\cdot +T]$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0, $T$</td>
<td>$[E \rightarrow T\cdot], [T \rightarrow T\cdot \times F]$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0, $F$</td>
<td>$[T \rightarrow F\cdot]$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1, $E$</td>
<td>$[F \rightarrow (E\cdot)]$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3, +</td>
<td>$[E \rightarrow E+\cdot T]$</td>
<td>$T, F$</td>
</tr>
<tr>
<td>8</td>
<td>4, $\times$</td>
<td>$[T \rightarrow T\times \cdot F]$</td>
<td>$F$</td>
</tr>
<tr>
<td>9</td>
<td>6, )</td>
<td>$[F \rightarrow (E) \cdot]$</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7, $T$</td>
<td>$[E \rightarrow E+T\cdot], [T \rightarrow T\cdot \times F]$</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>8, $F$</td>
<td>$[T \rightarrow T\times F\cdot]$</td>
<td></td>
</tr>
</tbody>
</table>

Goddard 10b: 11
**Example: Constructing the Table**

Each of the above transitions produces a shift in the table.

For example, suppose in state 3 one considers the transition for \(+\). This defines a new item-set that produces state 7. A shift \(s_7\) is added to the table.
Determining the reductions seems obvious: take each completed item and make that a reduction. 

*But*, all our calculations are trying to answer:* when to reduce and when not to.*
We need FOLLOW sets. For variable $A$, $\text{FOLLOW}(A)$ is the set of all possible terminals that can follow $A$ in a string derived from $S$.

One includes eos symbol $\triangle$ in $\text{FOLLOW}(A)$ if variable $A$ can occur at the end of a derived string.

For example, if $S \Rightarrow^{*} BCxBAyxxC$, then one adds $y$ to $\text{FOLLOW}(A)$ and $\triangle$, $x$ to $\text{FOLLOW}(C)$. 
Example: FOLLOW sets

In the arithmetic grammar, FOLLOW(E) is \{ \), \(, \Delta, + \}. Both FOLLOW(T) and FOLLOW(F) are \{ \), \(, \Delta, +, \times \}.

For instance, \(E' \rightarrow E \rightarrow T \rightarrow T \times F \rightarrow F \times F\) shows that \(\times \in \text{FOLLOW}(F)\).
Adding the Reductions to the Table

For each completed item, say an A-item, add reduction to table for all symbols that occur in FOLLOW(A). Except that, the reduction corresponding to artificial production $E' \rightarrow E$ becomes the acceptance point.
This process sounds rather daunting. Fortunately there is software to do it. The UNIX utility YACC generates LR tables for certain CFGs.
Practice

For the earlier grammar

1: \( S \rightarrow rL \)
2: \( L \rightarrow L, I \)
3: \( L \rightarrow I \)
4: \( I \rightarrow v \)

(a) Verify that \( \text{FOLLOW}(S) = \{ \Delta \} \) and \( \text{FOLLOW}(L) = \text{FOLLOW}(I) = \{ , , \Delta \} \).

(b) Determine the item-sets.

(c) Verify the entries in the table earlier (repeated next).
<table>
<thead>
<tr>
<th>State</th>
<th>Progress</th>
<th>( r )</th>
<th>( v )</th>
<th>eos</th>
<th>( S )</th>
<th>( I )</th>
<th>( L )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>s1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( r )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>( S )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>acc</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( v )</td>
<td></td>
<td>R4</td>
<td></td>
<td></td>
<td>R4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( I )</td>
<td></td>
<td>R3</td>
<td></td>
<td></td>
<td>R3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( rL )</td>
<td></td>
<td>s6</td>
<td></td>
<td></td>
<td>R1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>( L, )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>( L, I )</td>
<td></td>
<td>R2</td>
<td></td>
<td></td>
<td>R2</td>
<td></td>
</tr>
</tbody>
</table>
### (b) State

<table>
<thead>
<tr>
<th>State</th>
<th>Contents of item-set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>([S' \rightarrow \cdot S], [S \rightarrow \cdot rL])</td>
</tr>
<tr>
<td>1</td>
<td>([S \rightarrow r \cdot L], [L \rightarrow \cdot L, I], [L \rightarrow \cdot I], [I \rightarrow \cdot v])</td>
</tr>
<tr>
<td>2</td>
<td>([S' \rightarrow S \cdot])</td>
</tr>
<tr>
<td>3</td>
<td>([I \rightarrow v \cdot])</td>
</tr>
<tr>
<td>4</td>
<td>([L \rightarrow I \cdot])</td>
</tr>
<tr>
<td>5</td>
<td>([S \rightarrow rL \cdot], [L \rightarrow L \cdot, I])</td>
</tr>
<tr>
<td>6</td>
<td>([L \rightarrow L, \cdot I], [I \rightarrow \cdot v])</td>
</tr>
<tr>
<td>7</td>
<td>([L \rightarrow L, I \cdot])</td>
</tr>
</tbody>
</table>
Summary

The states of the parser correspond to item-sets, generated by trying all possible next terminals. The reductions occur for completed items based on FOLLOW sets.