Let’s try building an SLR parsing table for another simple grammar:

\[
\begin{align*}
S & \rightarrow XaY \mid Y \\
X & \rightarrow bY \mid c \\
Y & \rightarrow X
\end{align*}
\]

Canonical collection:

I₀: \{S' \rightarrow \cdot, S \rightarrow \cdot XaY, S \rightarrow \cdot Y, X \rightarrow bY, X \rightarrow \cdot c, Y \rightarrow \cdot X\}

I₁: \{S' \rightarrow \cdot S\}

I₂: \{S \rightarrow \cdot Xa, Y \rightarrow \cdot X\}

I₃: \{S \rightarrow \cdot Y\}

I₄: \{X \rightarrow b \cdot Y, Y \rightarrow \cdot X, X \rightarrow \cdot bY, X \rightarrow \cdot c\}

I₅: \{X \rightarrow \cdot c\}

I₆: \{S \rightarrow Xa \cdot Y, Y \rightarrow \cdot X, X \rightarrow \cdot bY, X \rightarrow \cdot c\}

I₇: \{X \rightarrow bY\cdot\}

I₈: \{Y \rightarrow X\cdot\}

I₉: \{S \rightarrow XaY\cdot\}
\[
\begin{align*}
0 & \quad S' \rightarrow S \\
1, 2 & \quad S \rightarrow XaY \mid Y \\
3, 4 & \quad X \rightarrow bY \mid c \\
5 & \quad Y \rightarrow X
\end{align*}
\]

\begin{align*}
I_0: & \{S' \rightarrow \cdot, S \rightarrow \cdot XaY, S \rightarrow \cdot Y, X \rightarrow \cdot bY, X \rightarrow \cdot c, Y \rightarrow \cdot X\} \\
I_1: & \{S' \rightarrow S\cdot\} \\
I_2: & \{S \rightarrow X \cdot aY, Y \rightarrow X \cdot\} \\
I_3: & \{S \rightarrow Y \cdot\} \\
I_4: & \{X \rightarrow b \cdot Y, Y \rightarrow \cdot X, X \rightarrow \cdot bY, X \rightarrow \cdot c\} \\
I_5: & \{X \rightarrow c \cdot\} \\
I_6: & \{S \rightarrow XaY \cdot, Y \rightarrow X \cdot\} \\
I_7: & \{X \rightarrow bY \cdot\} \\
I_8: & \{Y \rightarrow X \cdot\} \\
I_9: & \{S \rightarrow XaY \cdot\}
\end{align*}

FOLLOW(S) = \{$\$\} \quad \text{FOLLOW(X)} = \{a, \$\}

FOLLOW(Y) = \{a, \$\}

\begin{tabular}{|c|c|c|c|c|c|}
\hline
STATE & action & goto & & & \\
\hline
0   & a & b & c & $ & S & X & Y \\
1   &   &   &   &   &   &   &   \\
2   &   &   &   &   &   &   &   \\
3   &   &   &   &   &   &   &   \\
4   &   &   &   &   &   &   &   \\
5   &   &   &   &   &   &   &   \\
6   &   &   &   &   &   &   &   \\
7   &   &   &   &   &   &   &   \\
8   &   &   &   &   &   &   &   \\
9   &   &   &   &   &   &   &   \\
\hline
\end{tabular}
Let's try a parse:

<table>
<thead>
<tr>
<th>STACK</th>
<th>INPUT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>cac$</td>
<td>s5</td>
</tr>
<tr>
<td>0c5</td>
<td>ac$</td>
<td>r4</td>
</tr>
<tr>
<td>0X2</td>
<td>ac$</td>
<td>s6, not r5!</td>
</tr>
<tr>
<td>0X2a6</td>
<td>c$</td>
<td>s5</td>
</tr>
<tr>
<td>0X2a6c5</td>
<td>$</td>
<td>r4</td>
</tr>
<tr>
<td>0X2a6X8</td>
<td>$</td>
<td>r5</td>
</tr>
<tr>
<td>0X2a6Y9</td>
<td>$</td>
<td>r1</td>
</tr>
<tr>
<td>0S1</td>
<td>$</td>
<td>acc</td>
</tr>
</tbody>
</table>

Let's try the alternative...

<table>
<thead>
<tr>
<th>STACK</th>
<th>INPUT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>cac$</td>
<td>s5</td>
</tr>
<tr>
<td>0c5</td>
<td>ac$</td>
<td>r4</td>
</tr>
<tr>
<td>0X2</td>
<td>ac$</td>
<td>try r5, instead of s6</td>
</tr>
<tr>
<td>0Y3</td>
<td>ac$</td>
<td>r2</td>
</tr>
<tr>
<td>0S1</td>
<td>ac$</td>
<td>error</td>
</tr>
</tbody>
</table>

In LR parsing, the stack should always contain a viable prefix, and the stack plus lookahead (unless it’s $) should be a prefix of a right-sentential form. Otherwise the parse will fail...
If we have $X$ on the stack, with lookahead $a$, we had better not reduce $X$ to $Y$: $Ya$ is not a prefix of a right-sentential form, despite the fact that $a \in \text{FOLLOW}(Y)$.

$$ S \Rightarrow XaY \Rightarrow bY aY $$

To get an $a$ we must use production $S \rightarrow XaY$. Then to get a $Y$ in front of that $a$, we must produce a $Y$ from the $X$, which we can do only with production $X \rightarrow bY$. (So we can’t get substring $Ya$ without a leading $b$.)

(Notice that in this case we showed that $Ba$ is not a prefix of any sentential form — of course it follows that it is not a prefix of any right-sentential form.)

But maybe there are still cases in which, in state 2 on lookahead $a$, the parser should reduce with production 5...

---

The SLR parsing table isn’t adequate...
The grammar in the previous example is not ambiguous, and can be parsed by the LR method, if only we can construct a more adequate parsing table.

We'll look first at canonical LR parsing tables. Then LALR parsing tables, which are smaller and weaker (and are what yacc builds!). Both methods can handle the previous examples.

We begin by enriching the notion of an item, to better take into account lookahead tokens.

**Definition** An LR(1) item is a pair of an LR(0) item and a token or $\$, typically written

$$[X \rightarrow \alpha \cdot \beta, a] .$$

Intuitively, the lookahead $a$ has no effect if $\beta \neq \epsilon$, but if $\beta = \epsilon$, then $[X \rightarrow \alpha \cdot \beta, a]$ calls for reduction by $X \rightarrow \alpha\beta$ if the lookahead token is $a$.

Thus, we will reduce by a production $X \rightarrow \alpha$ only on those lookaheads $a$ for which there is an item $[X \rightarrow \alpha\cdot, a]$.

The set of such $\alpha$’s is always a subset of FOLLOW($X$).

**LR(1) closure function**

**Definition** Given a set $I$ of LR(1) items from an augmented grammar, $\text{closure}(I)$ is the least set of items satisfying the following two conditions:

1. $I \subseteq \text{closure}(I)$
2. If $[X \rightarrow \alpha \cdot Y \beta, a] \in \text{closure}(I)$ and there is an LR(1) item $[Y \rightarrow \cdot \gamma, b]$ s.t. $b \in \text{FIRST}(\beta a)$ then $[Y \rightarrow \cdot \gamma, b] \in \text{closure}(I)$.

Claim: $\text{FIRST}(\beta a) \subseteq \text{FOLLOW}(Y)$. (As elsewhere, I suspect this is not true, but it is suggestive. Why not true? Because the definition of FOLLOW is about existence of sentential forms, but this definition is not.)
1. $I \subseteq \text{closure}(I)$

2. If $[X \rightarrow \alpha \cdot Y \beta, a] \in \text{closure}(I)$ and there is an LR(1) item $[Y \rightarrow \cdot \gamma, b] \text{ s.t. } b \in \text{FIRST}(\beta a)$, then $[Y \rightarrow \cdot \gamma, b] \in \text{closure}(I)$.

Example What is $\text{closure}([S' \rightarrow \cdot S, \$])$?

So we consider LR(1) items with first components $S \rightarrow \cdot X a Y$ and $S \rightarrow \cdot Y$. In this case, $\beta = \epsilon$, so $\text{FIRST}(\beta) = \{\$\}$.

So we add LR(1) items $[S \rightarrow \cdot X a Y, \$]$ and $[S \rightarrow \cdot Y, \$]$. Now consider $X \rightarrow \cdot b Y$ and $X \rightarrow \cdot c$. Here $\beta = a Y$, so $\text{FIRST}(\beta) = \{a\}$. Thus we add items $[X \rightarrow \cdot b Y, a]$ and $[X \rightarrow \cdot c, a]$.

Now consider $Y \rightarrow \cdot X$, with $\beta = \epsilon$. Since $\text{FIRST}(\beta) = \{\$\}$, we add item $[Y \rightarrow \cdot X, \$]$. Now consider $X \rightarrow \cdot b Y$ and $X \rightarrow \cdot c$, with $\beta = \epsilon$. Since $\text{FIRST}(\beta) = \{\$\}$, we add items $[X \rightarrow \cdot b Y, \$]$ and $[X \rightarrow \cdot c, \$]$. What is $\text{closure}([S \rightarrow X a \cdot Y, \$])$?

Example What is $\text{closure}([S \rightarrow X a Y, \$], [Y \rightarrow X, \$])$?

What is $\text{closure}([X \rightarrow b \cdot Y, a[\$]])$? (stands for 2 LR(1) items)

Consider LR(1) items with first component $Y \rightarrow \cdot X$. In this case, $\beta = \epsilon$. Since $\text{FIRST}(\beta) = \{\$\}$, we add $[Y \rightarrow \cdot X, \$]$, and since $\text{FIRST}(\beta a) = \{a\}$, we add $[Y \rightarrow \cdot X, a]$.

Now consider $X \rightarrow \cdot b Y$ and $X \rightarrow \cdot c$. Here again $\beta = \epsilon$, so $\text{FIRST}(\beta) = \{\$\}$ and $\text{FIRST}(\beta a) = \{a\}$. Thus we add $[X \rightarrow \cdot b Y, a[\$] and $[X \rightarrow \cdot c, a[\$]$. What is $\text{closure}([S \rightarrow X a \cdot Y, \$])$?
**LR(1) goto function**

**Definition** For any set $I$ of LR(1) items and grammar symbol $X$, $\text{goto}(I, X)$ is the closure of the set of all LR(1) items

$$[A \to \alpha X \cdot \beta, a] \quad \text{s.t.} \quad [A \to \alpha \cdot X \beta, a] \in I.$$

**Example**

$I_0$: $\{[S' \to \cdot S, \$], [S \to \cdot XaY, \$], [S \to \cdot Y, \$]$

$[X \to \cdot bY, a|\$, [X \to \cdot c, a|\$, [Y \to \cdot X, \$]$

$\text{goto}(I_0, S) = \text{closure}([S' \to S', \$])$

$= ([S' \to S', \$])$

$\text{goto}(I_0, X) = \text{closure}([S \to X \cdot aY, \$, [Y \to X \cdot Y, \$])$

$= ([S \to X \cdot aY, \$, [Y \to X \cdot Y, \$])$

$\text{goto}(I_0, Y) = \text{closure}([S \to Y \cdot Y, \$])$

$= ([S \to Y \cdot Y, \$])$

$\text{goto}(I_0, b) = \text{closure}([X \to b \cdot Y, a|\$])$

$= ([X \to b \cdot Y, a|\$, [Y \to X, a|\$, [X \to bY, a|\$, [X \to \cdot c, a|\$]$

$\text{goto}(I_0, c) = \text{closure}([X \to c \cdot, a|\$])$

$= ([X \to c \cdot, a|\$])$

---

**Construction of sets of LR(1) items**

Here is the algorithm for construction of the canonical collection of sets of LR(1) items for an augmented grammar:

$$C := \{ \text{closure}([S' \to \cdot S, \$]) \};$$

repeat

for each set of LR(1) items $I \in C$ and each grammar symbol $X$

s.t. $\text{goto}(I, X)$ is not empty and is not already in $C$
do

add $\text{goto}(I, X)$ to $C$

until no more sets of items can be added to $C$ in this way

So this is the same construction we used for LR(0) items — but now for LR(1) items, with the LR(1) versions of closure and goto, and the initial LR(1) item.
Construction of canonical LR(1) parsing table

1. Construct the canonical collection $C = \{I_0, \ldots, I_n\}$ of sets of LR(0) items. Each element of $C$ is a “state”, and we’ll often write $k$ to refer to the state $I_k$.

2. The applicable parsing actions for state $k$ and lookahead $a$ are determined as follows:
   a) If $\text{goto}(k, a) = j$, then “shift and go to state $j$” is applicable.
   b) If $[B \rightarrow \alpha \cdot a] \in k$ and $B$ is not $S'$, then “reduce by production $B \rightarrow \alpha$” is applicable.
   c) If $[S' \rightarrow S \cdot ] \in k$ and $a = \$$, then “accept” is applicable.

If any conflicting actions are generated in step 2, the grammar is “not LR(1)”.

3. For all states $k$ and variables $A$, if $\text{goto}(k, A)$ is nonempty, then $\text{goto}(k, A)$ is the corresponding goto entry.

4. All remaining undefined entries have action “error”.

5. If $I_k = \text{closure}([S' \rightarrow \cdot S, \$$])$, then $k$ is the initial state of the parser.