4.6.4 Constructing SLR-Parsing Tables

The SLR method for constructing parsing tables is a good starting point for studying LR parsing. We shall refer to the parsing table constructed by this method as an SLR table, and to an LR parser using an SLR-parsing table as an SLR parser. The other two methods augment the SLR method with lookahead information.

The SLR method begins with LR(0) items and LR(0) automata, introduced in Section 4.5. That is, given a grammar, G, we augment G to produce G', with a new start symbol S'. From G', we construct C, the canonical collection of sets of items for G' together with the GOTO function.

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		**id** $*$ **id** $+$ **id** \$	shift
(2)	0 5	**id**	$*$ **id** $+$ **id** \$	reduce by $F \rightarrow$ **id**
(3)	0 3	F	$*$ **id** $+$ **id** \$	reduce by $T \rightarrow F$
(4)	0 2	T	$*$ **id** $+$ **id** \$	shift
(5)	0 2 7	$T*$	**id** $+$ **id** \$	shift
(6)	0 2 7 5	$T*$ **id**	$+$ **id** \$	reduce by $F \rightarrow$ **id**
(7)	0 2 7 10	$T*F$	$+$ **id** \$	reduce by $T \rightarrow T*F$
(8)	0 2	T	$+$ **id** \$	reduce by $E \rightarrow T$
(9)	0 1	E	$+$ **id** \$	shift
(10)	0 1 6	$E+$	**id** \$	shift
(11)	0 1 6 5	$E+$ **id**	\$	reduce by $F \rightarrow$ **id**
(12)	0 1 6 3	$E+F$	\$	reduce by $T \rightarrow F$
(13)	0 1 6 9	$E+T$	\$	reduce by $E \rightarrow E+T$
(14)	0 1	E	\$	accept

Figure 4.38: Moves of an LR parser on **id** $*$ **id** $+$ **id**

The ACTION and GOTO entries in the parsing table are then constructed using the following algorithm. It requires us to know FOLLOW(A) for each nonterminal A of a grammar (see Section 4.4).

Algorithm 4.46: Constructing an SLR-parsing table.

INPUT: An augmented grammar G'.

OUTPUT: The SLR-parsing table functions ACTION and GOTO for G'.

METHOD:

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for G'.

2. State i is constructed from I_i. The parsing actions for state i are determined as follows:

   (a) If $[A \rightarrow \alpha \cdot a\beta]$ is in I_i and GOTO$(I_i, a) = I_j$, then set ACTION$[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \rightarrow \alpha \cdot]$ is in I_i, then set ACTION$[i, a]$ to "reduce $A \rightarrow \alpha$" for all a in FOLLOW(A); here A may not be S'.

   (c) If $[S' \rightarrow S \cdot]$ is in I_i, then set ACTION$[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If GOTO$(I_i, A) = I_j$, then GOTO$[i, A] = j$.

4. All entries not defined by rules (2) and (3) are made "error."

5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S]$.

□

The parsing table consisting of the ACTION and GOTO functions determined by Algorithm 4.46 is called the *SLR(1) table for G*. An LR parser using the SLR(1) table for G is called the SLR(1) parser for G, and a grammar having an SLR(1) parsing table is said to be *SLR(1)*. We usually omit the "(1)" after the "SLR," since we shall not deal here with parsers having more than one symbol of lookahead.

Example 4.47: Let us construct the SLR table for the augmented expression grammar. The canonical collection of sets of LR(0) items for the grammar was shown in Fig. 4.31. First consider the set of items I_0:

$$
\begin{aligned}
&E' \to \cdot E \\
&E \to \cdot E + T \\
&E \to \cdot T \\
&T \to \cdot T * F \\
&T \to \cdot F \\
&F \to \cdot (E) \\
&F \to \cdot \mathbf{id}
\end{aligned}
$$

The item $F \to \cdot(E)$ gives rise to the entry ACTION$[0, ($] = shift 4, and the item $F \to \cdot\mathbf{id}$ to the entry ACTION$[0, \mathbf{id}]$ = shift 5. Other items in I_0 yield no actions. Now consider I_1:

$$
\begin{aligned}
&E' \to E\cdot \\
&E \to E\cdot + T
\end{aligned}
$$

The first item yields ACTION$[1, \$]$ = accept, and the second yields ACTION$[1, +]$ = shift 6. Next consider I_2:

$$
\begin{aligned}
&E \to T\cdot \\
&T \to T\cdot * F
\end{aligned}
$$

Since FOLLOW$(E) = \{\$, +,)\}$, the first item makes

$$\text{ACTION}[2, \$] = \text{ACTION}[2, +] = \text{ACTION}[2,)] = \text{reduce } E \to T$$

The second item makes ACTION$[2, *]$ = shift 7. Continuing in this fashion we obtain the ACTION and GOTO tables that were shown in Fig. 4.31. In that figure, the numbers of productions in reduce actions are the same as the order in which they appear in the original grammar (4.1). That is, $E \to E + T$ is number 1, $E \to T$ is 2, and so on. □

Example 4.48: Every SLR(1) grammar is unambiguous, but there are many unambiguous grammars that are not SLR(1). Consider the grammar with productions

$$\begin{array}{lcl} S & \rightarrow & L = R \mid R \\ L & \rightarrow & *R \mid \mathbf{id} \\ R & \rightarrow & L \end{array} \tag{4.49}$$

Think of L and R as standing for l-value and r-value, respectively, and $*$ as an operator indicating "contents of."[5] The canonical collection of sets of LR(0) items for grammar (4.49) is shown in Fig. 4.39.

I_0: $S' \rightarrow \cdot S$
$S \rightarrow \cdot L = R$
$S \rightarrow \cdot R$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$
$R \rightarrow \cdot L$

I_1: $S' \rightarrow S\cdot$

I_2: $S \rightarrow L\cdot = R$
$R \rightarrow L\cdot$

I_3: $S \rightarrow R\cdot$

I_4: $L \rightarrow *\cdot R$
$R \rightarrow \cdot L$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$

I_5: $L \rightarrow \mathbf{id}\cdot$

I_6: $S \rightarrow L = \cdot R$
$R \rightarrow \cdot L$
$L \rightarrow \cdot * R$
$L \rightarrow \cdot \mathbf{id}$

I_7: $L \rightarrow *R\cdot$

I_8: $R \rightarrow L\cdot$

I_9: $S \rightarrow L = R\cdot$

Figure 4.39: Canonical LR(0) collection for grammar (4.49)

Consider the set of items I_2. The first item in this set makes ACTION$[2, =]$ be "shift 6." Since FOLLOW(R) contains $=$ (to see why, consider the derivation $S \Rightarrow L = R \Rightarrow *R = R$), the second item sets ACTION$[2, =]$ to "reduce $R \rightarrow L$." Since there is both a shift and a reduce entry in ACTION$[2, =]$, state 2 has a shift/reduce conflict on input symbol $=$.

Grammar (4.49) is not ambiguous. This shift/reduce conflict arises from the fact that the SLR parser construction method is not powerful enough to remember enough left context to decide what action the parser should take on input $=$, having seen a string reducible to L. The canonical and LALR methods, to be discussed next, will succeed on a larger collection of grammars, including

[5] As in Section 2.8.3, an l-value designates a location and an r-value is a value that can be stored in a location.

grammar (4.49). Note, however, that there are unambiguous grammars for which every LR parser construction method will produce a parsing action table with parsing action conflicts. Fortunately, such grammars can generally be avoided in programming language applications. □