Building LR Tables

Recall the definition of a context-free grammar

$$\mathcal{G} \stackrel{\text{def}}{=} \langle V_I, V_N, V_G, \Pi \rangle \tag{1}$$

where

$$\begin{array}{rcl} V & \stackrel{\text{def}}{=} & V_I \cup V_N \\ V_I \cap V_N & \stackrel{\text{def}}{=} & \{\} \\ V_G & \stackrel{\text{def}}{\subseteq} & V_N \\ \Pi & \stackrel{\text{def}}{\subseteq} & V_N \times V^* \end{array}$$

A rewriting

$$\alpha \mathrm{B} \gamma \leftarrow \alpha \beta \gamma$$

is allowed if

$$\mathrm{B} \leftarrow \beta \in \Pi$$

and is written to show the explicit CFG rule application as

$$\alpha \mathrm{B} \gamma \stackrel{\mathrm{B} \leftarrow \beta}{\leftarrow} \alpha \beta \gamma$$

The *language* defined by a CFG $\mathcal{G}$ is the set of strings of input symbols that can be reduced to a goal.

$$\mathcal{L}(\mathcal{G}) \stackrel{\text{def}}{=} \{\alpha \mid G \leftarrow^* \alpha) \wedge G \in V_G\} \cap V_I^*$$

Text $\tau \in \mathcal{L}(\mathcal{G})$ (is syntactically correct) if

$$\mathrm{G} \stackrel{\rho}{\leftarrow} \tau$$

where

$$\rho \in \Pi^*$$

The following operational definition of parsing forces a left-to-right order on the rule applications

$$\begin{array}{rcl} G \in V_G & \Rightarrow & \mathcal{C}(G, \lambda) \\ B \leftarrow \beta \in \Pi \wedge \mathcal{C}(\sigma B, \tau) & \Rightarrow & \mathcal{C}(\sigma \beta, \tau) \\ a \in V_I \wedge \mathcal{C}(\sigma \mathrm{a}, \tau) & \Rightarrow & \mathcal{C}(\sigma, \mathrm{a} \tau) \\ \mathcal{C}(\lambda, \tau) & \Rightarrow & \tau \in \mathcal{L}(\mathcal{G}) \end{array}$$

Suppose we have a sequence of $n+1$ left-to-right rewritings (the application of the second rule in the definition above)

$$\mathrm{G} \overset{r_n}{\leftarrow} \sigma_n \mathrm{A}_n \tau_n \ldots \overset{r_2}{\leftarrow} \sigma_2 \mathrm{A}_2 \tau_2 \overset{r_1}{\leftarrow} \sigma_1 \mathrm{A}_1 \tau_1 \overset{r_0}{\leftarrow} \tau_0$$

where

$$\begin{array}{rcl} \tau_i & \in & V_I^* \\ \mathrm{A}_i & \in & V_N \\ \sigma_i & \in & V^* \\ r_i & \in & \Pi \end{array}$$

then $\rho = r_n \ldots r_2 r_1 r_0$ is a left-to-right parse of τ_0, and the set of values $\sigma_i \mathrm{A}_i$ are the values of the parse stack just after a substituion has been applied.

Consider, for a fixed $\mathcal{G}$, the set of all such parse stacks. The set is a language. It can be described by a new CFG $\mathcal{G}'$ in which the phrase names are designated by the notation [text].

$$\begin{array}{rcl} V_I' & \overset{\text{def}}{=} & V \\ V_N' & \overset{\text{def}}{=} & \{[\mathrm{A} \leftarrow \alpha] \mid \mathrm{A} \leftarrow \alpha\beta \in \Pi\} \\ V_G' & \overset{\text{def}}{=} & \Pi \\ \Pi' & \overset{\text{def}}{=} & \{[\mathrm{A} \leftarrow \alpha\mathrm{B}] \leftarrow [\mathrm{A} \leftarrow \alpha]\mathrm{B} \mid \mathrm{B} \in V \wedge \mathrm{A} \leftarrow \alpha\mathrm{B}\gamma \in \Pi\} \\ & \cup & \{[\mathrm{A} \leftarrow \alpha\mathrm{B}] \leftarrow [\mathrm{B} \leftarrow \lambda] \mid \mathrm{B} \in V_N \wedge \mathrm{A} \leftarrow \alpha\mathrm{B}\gamma \in \Pi\} \\ & \cup & \{[\mathrm{A} \leftarrow \alpha] \leftarrow \lambda \mid \mathrm{A} \leftarrow \alpha \in \Pi\} \\ \mathcal{G}' & \overset{\text{def}}{=} & \langle V_I', V_N', \mathrm{G}', \Pi' \rangle \end{array}$$

$\mathcal{G}'$ is a finite automaton. The start state is $[\mathrm{G} \leftarrow \lambda]$. If one applies the NFA to DFA transformation, the resulting DFA can walk the catenaton of the parse stack and input text, reaching a final state when a rule can be applied.

This is the basis of the LR technology. The LR automaton walks the parse stack, pulling new text off the as yet unprocessed input as necessary. In a final state the automaton is temporarily abandoned, the newly identified reduction is applied to the top of the parse stack, and the process starts over at the bottom of the parse stack.

LR provides an implementation of the operational definition of parsing given above. The tricky part is that more than one final state can sometimes be reached by the DFA. Resolving the ambiguous situations with k-symbol lookahead is what LR(k) signifies.

There is an LR(0) DFA for any CFG. There are no languages of interest that can be deterministically parsed using an LR(0) machine; at least LR(1) is necessary. It turns out that the LR(1) DFA is large enough raise concerns for memory occupancy of the LR(1) tables. Much research has gone into reducing the table size.

As a practical matter, LR(1) is also enough. Computer language designers restrict their languages to comply. Both the C and Java designers had some difficulty. A meta-rule has been applied: the reduction that reaches furthest into the input text is chosen (greedy shift).

Knuth's algorithm combines the construction of the DFA and the transformation to NFA. The LR NFA is represented by a set of sets of marked rules with lookahead. The marked rules correspond to the partial rules in the NFA above. The lookahead is a set of symbols carried along with the marked rules. The idea is that a final state can be ignored if there is no correct continuation after the application of the rule; that is, if the restarted parse after the rule is applied will inevitably run into an error.

The algorithm to build the LR(1) machine alternates two computations: closure and shifts. After closure a set corresponds to a state of the LR(1) DFA. Each shift corresponds to a shift of the LR(1) DFA. The starting position is

$$\{\mathrm{G} \leftarrow .\alpha\} : \{\}$$

that is the leftmost rule marked in the leftmost place with no lookahead. Marked rules must be added for each phrase name immediately the right of the mark. The lookahead is the heads of the text immediately to the right of the phrase name (in the rule in which it was found). For LR(1), the lookahead strings have length 1. The process of closure is continued recursively until no additional marked rules are found.

For each symbol immediately to the right of the mark, a new set is created containing the marked rules with the mark moved past the symbol. Then each of the new sets must be closed.

When there are no new sets. the LR machine is complete.