4.6.3 The LR-Parsing Algorithm

A schematic of an LR parser is shown in Fig. 4.35. It consists of an input, an output, a stack, a driver program, and a parsing table that has two parts (ACTION and GOTO). The driver program is the same for all LR parsers; only the parsing table changes from one parser to another. The parsing program reads characters from an input buffer one at a time. Where a shift-reduce parser would shift a symbol, an LR parser shifts a *state*. Each state summarizes the information contained in the stack below it.

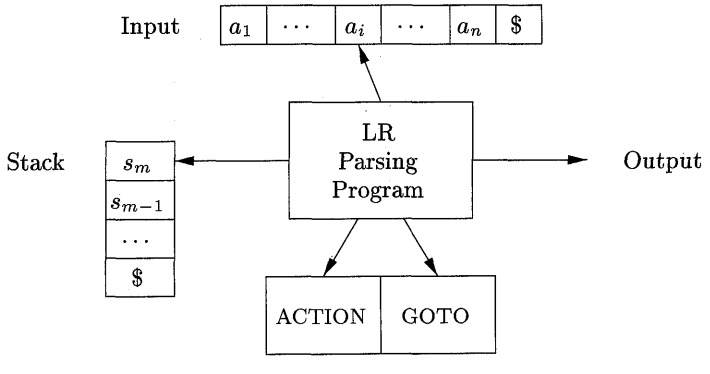

Figure 4.35: Model of an LR parser

The stack holds a sequence of states, $s_0 s_1 \cdots s_m$, where s_m is on top. In the SLR method, the stack holds states from the LR(0) automaton; the canonical-LR and LALR methods are similar. By construction, each state has a corresponding grammar symbol. Recall that states correspond to sets of items, and that there is a transition from state i to state j if GOTO$(I_i, X) = I_j$. All transitions to state j must be for the same grammar symbol X. Thus, each state, except the start state 0, has a unique grammar symbol associated with it.[4]

[4]The converse need not hold; that is, more than one state may have the same grammar

Structure of the LR Parsing Table

The parsing table consists of two parts: a parsing-action function ACTION and a goto function GOTO.

1. The ACTION function takes as arguments a state i and a terminal a (or \$, the input endmarker). The value of ACTION$[i, a]$ can have one of four forms:
   (a) Shift j, where j is a state. The action taken by the parser effectively shifts input a to the stack, but uses state j to represent a.
   (b) Reduce $A \to \beta$. The action of the parser effectively reduces β on the top of the stack to head A.
   (c) Accept. The parser accepts the input and finishes parsing.
   (d) Error. The parser discovers an error in its input and takes some corrective action. We shall have more to say about how such error-recovery routines work in Sections 4.8.3 and 4.9.4.
2. We extend the GOTO function, defined on sets of items, to states: if GOTO$[I_i, A] = I_j$, then GOTO also maps a state i and a nonterminal A to state j.