LR Parser

- LR(k) parser is a bottom-up parser

  L → left-to-right scanning of input

  R → constructing right most derivation in reverse

  k → number of input symbols of lookahead that are used to make parsing decision

  k = 0, k = 1 } are of practical interest.

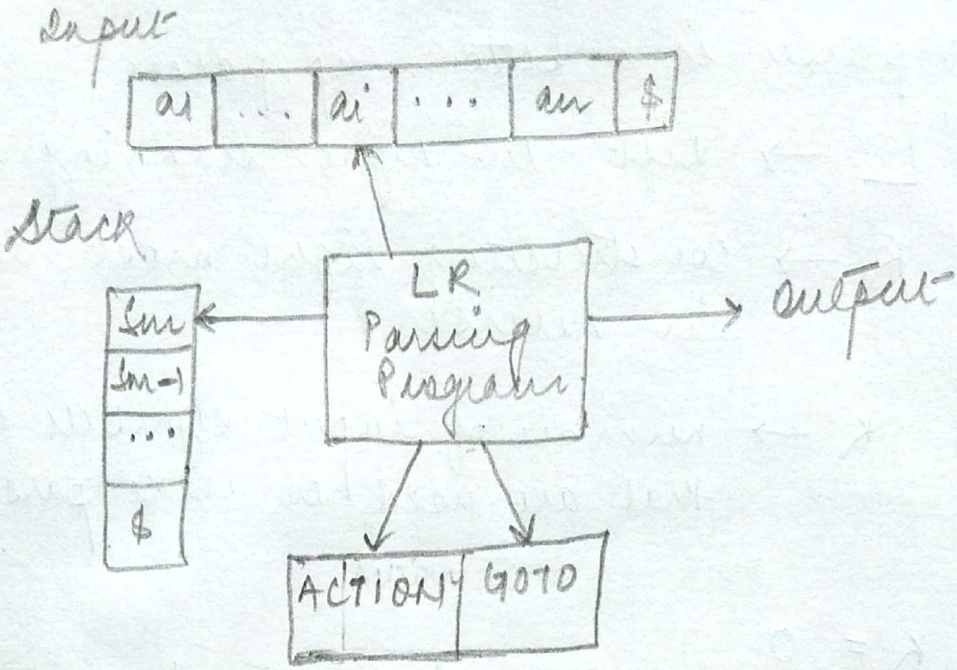
  when (k) is omitted, k is assumed to be 1.

- LR parser are table-driven
- the grammar for which parsing table can be constructed is said to be LR grammar.
- For a grammar to be LR, it is sufficient that a left-to-right shift-reduce parser be able to recognize handles of right-sentential forms when they on the top of the stack.

Benefits

- LR parser can be constructed to recognize all programming constructs for which CFG can be written
- No backtracking is performed.

Model of LR Parser



- consists of an input
  an output
  a stack [holds a sequence of states]
  a driver program [same for all LR Parsers]
  a parsing table [changes from one parser to another]

- Parsing program reads characters from input buffer one at a time.
- LR parser shifts a state. Each state summarizes the information contained in stack below it.

Structure of LR Parsing Table

The parsing table consists of two parts:

- A parsing action function ACTION
- A goto function GOTO.

1. The ACTION function takes as arguments a state i and a terminal a (or $, the input end marker). The value of ACTION[i, a] can have one of the four forms:

a) shift j, where j is a state: The parser shifts input 'a' to the stack but uses state j to represent 'a'.

b) Reduce $A \rightarrow \beta$: The action of parser effectively reduces β on the top of the stack to head A.

c) Accept: The parser accepts the input & finishes parsing.

d) Error: The parser discovers an error in its input & takes corrective action.

2. A GOTO function: If GOTO[I_i, A] = I_j, then GOTO maps a state i & a non-terminal A to state j.