CFG → PDA (computing model for CFL/CFG)

→ DFA of states & transitions & closure of rules

→ Build LR-Parse Table

→ LR-Parsing using LR-Parse Table

Algorithm to Build DFA Modeling LR Parsing

1. Create an empty queue named STATES.

2. Compute closure(S' → ・ S). This is state 0, the start state, in the DFA.

3. Put state 0 into STATES.

4. Repeat until STATES is empty.

   (a) Take any state p out of STATES for processing.

   (b) For each item A → v ・ xw in state p with A ∈ V, v and w in $(V \cup T)^*$ and x ∈ V or x ∈ T,

      i. Create a new state s with its set of items closure(A → vx ・ w).

      ii. If s's set of items is unique,

         A. Assign s a new state number in the DFA,

         BC. Add the transition δ(p, x) = s, and place s in STATES.

      iii. If state's set of items matches an existing state's set of items,

         A. Let r be the existing state equivalent to s,

         BC. Add the transition δ(p, x) = r, and discard state s.

5. For each state q in the DFA, if q has any items with the marker at the right end, such as A → w ・,

   then add q to F as a final state.

Algorithm to Build LR(1) Parse Table

1. For each state q,

   (a) for each transition labeled x from state q to state p do

      i. if x ∈ T then LR[q, x] = s p, where s and g means Shift and Go, respectively.

      ii. if x ∈ V then LR[q, x] = g p.

   (b) if state q is a final state with S → S ・, then LR[q, $] = acc

   (c) if state q is a final state with other completed marked productions,

      then for those A → w ・ in q numbered as N) A → w do:

         for each *a* in FOLLOW(A):

            LR[q, *a*] = r N. where r means Reduce.

2. All blank entries represent an error.

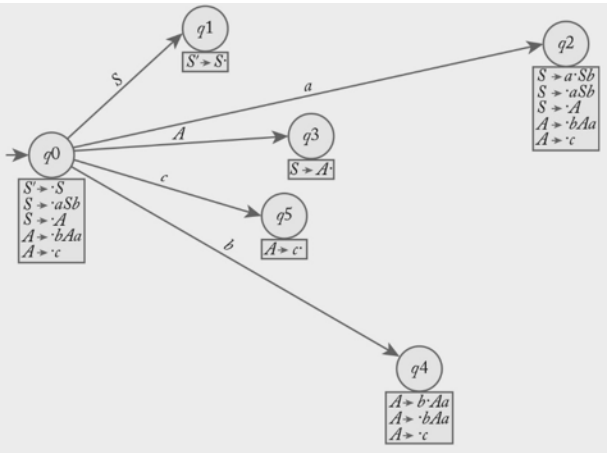
LR(1) Parsing Algorithm

```
state = 0
push(state)
lookahead = get()
entry = LR[state, lookahead]
while entry.action == s or entry.action == r:  // Shift or Reduce
        if entry.action == s then: {
                push(lookahead)
                lookahead = get()
                 state = entry.state
                push(state)  }
        else: {                                          // entry.action is r, i.e. Reduce
                 for size(entry.rhs)*2 do:
                        pop()
                state = top()
                push(entry.lhs)
                entry = LR[state, entry.lhs]
                state = entry.state
                push(state)  }
        entry = LR[state, lookahead]
if entry.action == acc and lookahead == $ then
         accept string
else:
        error
```

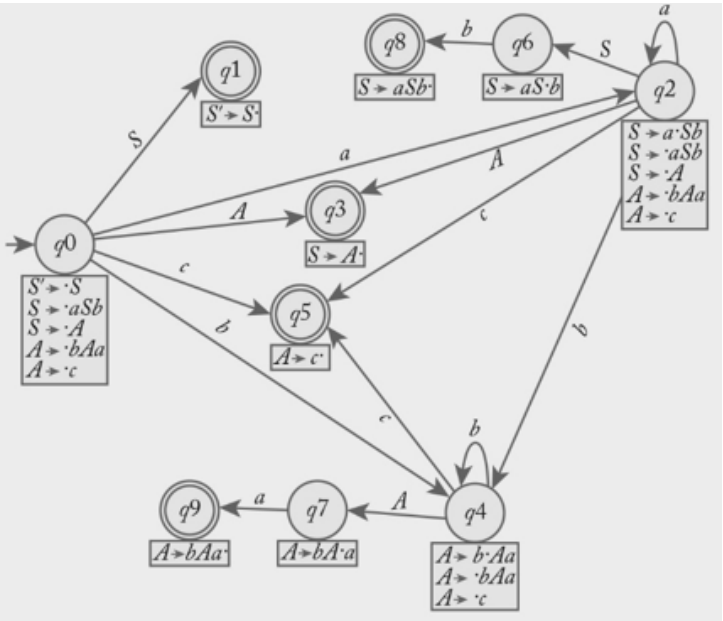
Example:

1) **CFG:** (0) S′ → S, (1) S → aSb, (2) S → A, (3) A → bAa, (4) A → c
2) **Closure(S' → ・S) → DFA** with the related states/transitions/closure(rule)

   = {S' → ・S, S → ・aSb, S → ・A, A → ・bAa, A → ・c}



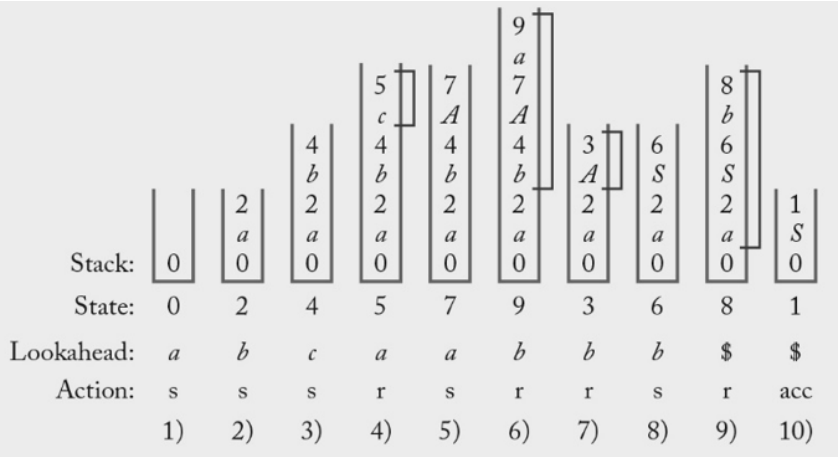
3) Closure(All of the rules) → complete DFA



4) LR(1) Parse Table

	ACTION				GOTO	
	a	***b***	***c***	***$***	***S***	***A***
0	s2	s4	s5		g1	g3
1				acc		
2	s2	s4	s5		g6	g3
3		r2		r2		
4		s4	s5			g7
5	r4	r4		r4		
6		s8				
7	s9					
8		r1		r1		
9	r3	r3		r3		

5) Trace String '*abcab*' with LR(1) Parse Table



Stack:	0	2 *a* 0	4 *b* 2 *a* 0	5 *c* 4 *b* 2 *a* 0	7 *A* 4 *b* 2 *a* 0	9 *a* 7 *A* 4 *b* 2 *a* 0	3 *A* 2 *a* 0	6 *S* 2 *a* 0	8 *b* 6 *S* 2 *a* 0	1 *S* 0
State:	0	2	4	5	7	9	3	6	8	1
Lookahead:	*a*	*b*	*c*	*a*	*a*	*b*	*b*	*b*	$	$
Action:	s	s	s	r	s	r	r	s	r	acc
	1)	2)	3)	4)	5)	6)	7)	8)	9)	10)

- Start in row 0 with lookahead *a*, push 0 on stack.
- LR[0, *a*] = s2 → shift *a* and 2 on stack, now lookahead = b.
- LR[2, b] = s4 → shift b and 4 on stack, now lookahead = c.
- LR[4, c] = s5 → shift c and 5 on stack, now lookahead = *a*.
- LR[5, *a*] = r4 → reduce by rule 4, A → c.
    - Pop two symbols off the stack (5 and c).
    - Current state = 4 (value on top of the stack).
    - Push A on the stack.
    - LR[4, A] = g7, push 7 on stack.
- LR[7, *a*] = s9 → shift a and 9 on stack, now lookahead = b.
- LR[9, b] = r3 → reduce by rule 3, A → b*A*a.
    - Pop 6 symbols off the stack (9, *a*, 7, A, 4, b).
    - Current state = 2 (value on top of the stack).
    - Push A on the stack.
    - LR[2, A] = g3 → push 3 on stack.
- ……..