* Trình phân tích cú pháp từ dưới lên bắt đầu từ các nút lá của cây phân tích cú pháp và hoạt động theo hướng đi lên cho đến khi nó đi đến nút gốc.
* Phân tích cú pháp Shift-Reduce: sử dụng 2 hành động để phân tích cú pháp từ dưới lên, đó là dịch shift và reduce.
  + shift: Đẩy ký hiệu vào stack. Ký hiệu được shift được coi như một nút duy nhất của cây phân tích cú pháp.
  + reduce: Thay thế a ở đỉnh ngăn xếp bằng non-terminal A với A → a. Nói cách khác, ta pop a, push A.
  + Chốt (handle) là các ký hiệu được nối với nhau ở mỗi bước phân tích cú pháp để biểu diễn bước tiếp theo của quá trình phân tích cú pháp.
* Trình phân tích cú pháp LR: là trình phân tích cú pháp không đệ quy, Shift-Reduce và từ dưới lên.
  + Nó sử dụng nhiều loại văn phạm phi ngữ cảnh, điều này làm cho nó trở thành kỹ thuật phân tích cú pháp hiệu quả nhất.
  + Nó còn được gọi là trình phân tích cú pháp LR(k), trong đó:
    - L (left-to-right): Duyệt chuỗi đầu vào từ trái sang phải.
    - R (right-most derivation): Xây dựng chuỗi dẫn xuất phải nhất.
    - k: Biểu thị số lượng ký hiệu lookahead để đưa ra quyết định. Nếu không đề cập đến, ta ngầm hiểu k = 1.
  + LR item: 1 luật đang được phân tích dở, sử dụng dấu . để ngăn giữa phần trước và phần sau
    - . ở trước 1 terminal: đề xuất cho shift.
    - . ở cuối bên phải: đề xuất cho reduce.
    - VD: Luật S → ABC sẽ có 4 item:
      * S → .ABC
      * S → A.BC
      * S → AB.C
      * S → ABC.
    - Basis và Closure: Cho I là tập các LR item, ta có thuật toán tính Closure của I như sau:

CLOSURE()

**repeat**

**for** any item in

**for** any production

**until** does not change

**return**

* + Theo dõi các trạng thái mà nó đã đi qua bằng cách đẩy các trạng thái đó vào stack cùng với các ký hiệu văn phạm.
  + action[s, a] với s là 1 trạng thái và a là 1 terminal, có thể nhận 1 trong 4 giá trị sau:
    - shift s: đẩy ký hiệu input và trạng thái s vào stack.
    - reduce A → a: reduce bởi luật sinh A → a, pop trạng thái s’ và a ở đầu stack ra và push lần lượt A và trạng thái s vào stack, trong đó s = goto[s’’, A]. Trong đó s’’ là trạng thái ở đầu stack sau khi đã pop s’.
    - accept: phân tích thành công.
    - error: phát hiện lỗi.
  + goto[s, A] với s là 1 trạng thái và A là 1 non-terminal, chỉ ra cách dịch chuyển trạng thái. Ta có thuật toán để tính goto(I, X) với I là tập các LR item và X là một ký hiệu văn phạm như sau:

GOTO():

= empty set

**for** any item in

**return** CLOSURE()

* Các thuật toán được sử dụng rộng rãi để xây dựng trình phân tích cú pháp LR:
  + LR(0): Thuật toán cơ bản, mọi thuật toán họ LR đều dựa trên nó.
  + SLR(1): Trình phân tích cú pháp LR đơn giản (simple LR), cải tiến một chút từ LR(0), mạnh hơn, triển khai đơn giản và nhanh chóng. Hoạt động trên lớp văn phạm nhỏ nhất, ít trạng thái do đó bảng nhỏ.
  + LR(1): Còn được gọi là LR chính tắc (Canonical LR), sử dụng cho nhiều loại văn phạm, kích thước bảng và số lượng các trạng thái lớn.
  + LALR(1): Trình phân tích cú pháp lookahead LR, cân bằng giữa SLR và LR, phổ biến nhất trong thực tế, đủ dùng cho hầu hết các văn phạm nhân tạo. Nó có cùng kích thước và tốc độ với SLR nhưng có thể giải quyết được nhiều xung đột hơn.
* Các thành viên của họ phân tích cú pháp LR đều sử dụng một automaton, đó là máy trạng thái hữu hạn đặc trưng (CFSM).

Ví dụ: Dựa vào bảng phân tích cú pháp SLR(1) cho ngôn ngữ máy tính trong sách. Ta phân tích cú pháp cho chương trình dưới đây:

read A

res := A + 1

write res

Ta phân tích như sau:

|  |  |  |
| --- | --- | --- |
| Stack | Input stream | Comment |
| 0 | read A… |  |
| 0 read 1 | A res… | action[0, read] = s1 |
| 0 | stmt res… | action[1, id(A)] = b5 |
| 0 | smtm\_list res… | action[0, stmt] = b3 |
| 0 stmt\_list 2 | res :=… | action[0, stmt\_list] = s2 |
| 0 stmt\_list 2 id 3 | := A… | action[2, id(res)] = s3 |
| 0 stmt\_list 2 id 3 := 5 | A + 1… | action[3, :=] = s5 |
| 0 stmt\_list 2 id 3 := 5 | factor + 1… | action[5, id(A)] = b12 |
| 0 stmt\_list 2 id 3 := 5 | term + 1… | action[5, factor] = b9 |
| 0 stmt\_list 2 id 3 := 5 term 7 | + 1… | action[5, term] = s7 |
| 0 stmt\_list 2 id 3 := 5 | expr + 1… | action[7, +] = r7 |
| 0 stmt\_list 2 id 3 := 5 expr 9 | + 1… | action[5, expr] = s9 |
| 0 stmt\_list 2 id 3 := 5 expr 9 | add\_op 1… | action[9, +] = b14 |
| 0 stmt\_list 2 id 3 := 5 expr 9 add\_op 10 | 1 write… | action[9, add\_op] = s10 |
| 0 stmt\_list 2 id 3 := 5 expr 9 add\_op 10 | factor write… | action[10, number(1)] = b13 |
| 0 stmt\_list 2 id 3 := 5 expr 9 add\_op 10 | term write… | action[10, factor] = b9 |
| 0 stmt\_list 2 id 3 := 5 expr 9 add\_op 10 term 13 | write res… | action[10, term] = s13 |
| 0 stmt\_list 2 id 3 := 5 | expr write res… | action[13, write] = r8 |
| 0 stmt\_list 2 id 3 := 5 expr 9 | write res… | action[5, expr] = s9 |
| 0 stmt\_list 2 | stmt write res… | action[9, write] = r4 |
| 0 | stmt\_list write res… | action[2, stmt] = b2 |
| 0 stmt\_list 2 | write res… | action[0, stmt\_list] = s2 |
| 0 stmt\_list 2 write 4 | res $$ | action[2, write] = s4 |
| 0 stmt\_list 2 write 4 | factor $$ | action[4, id(res)] = b12 |
| 0 stmt\_list 2 write 4 | term $$ | action[4, factor] = b9 |
| 0 stmt\_list 2 write 4 term 7 | $$ | action[4, term] = s7 |
| 0 stmt\_list 2 write 4 | expr $$ | action[7, $$] = r7 |
| 0 stmt\_list 2 write 4 expr 6 | $$ | action[4, expr] = s6 |
| 0 stmt\_list 2 | stmt $$ | action[6, $$] = r6 |
| 0 | stmt\_list $$ | action[2, stmt] = b2 |
| 0 stmt\_list 2 | $$ | action[0, stmt\_list] = s2 |
| 0 | program | action[2, $$] = b1 |
| [done] |  |  |