

Concepts of Programming Languages, CSCI 305, Fall 2021
Bottom-Up Parsing, Oct. 25

Bottom-Up Parsing, Section 2.3, pages 87-96

Hand-execute the SLR(1) parser using the table driven parsing code on the next page:

```
read A
read B
sum := A + B
write sum
write sum / 2
$$
```

Grammar:

1. $\text{program} \rightarrow \text{stmt_list } \$\$$
2. $\text{stmt_list} \rightarrow \text{stmt_list stmt}$
3. $\text{stmt_list} \rightarrow \text{stmt}$
4. $\text{stmt} \rightarrow \text{id} := \text{expr}$
5. $\text{stmt} \rightarrow \text{read id}$
6. $\text{stmt} \rightarrow \text{write expr}$
7. $\text{expr} \rightarrow \text{term}$
8. $\text{expr} \rightarrow \text{expr add_op term}$
9. $\text{term} \rightarrow \text{factor}$
10. $\text{term} \rightarrow \text{term mult_op factor}$
11. $\text{factor} \rightarrow (\text{expr})$
12. $\text{factor} \rightarrow \text{id}$
13. $\text{factor} \rightarrow \text{number}$
14. $\text{add_op} \rightarrow +$
15. $\text{add_op} \rightarrow -$
16. $\text{mult_op} \rightarrow *$
17. $\text{mult_op} \rightarrow /$

```

state = 1 .. number_of_states
symbol = 1 .. number_of_symbols
production = 1 .. number_of Productions
action_rec = record
    action : (shift, reduce, shift_reduce, error)
    new_state : state
    prod : production

parse_tab : array [symbol, state] of action_rec
prod_tab : array [production] of record
    lhs : symbol
    rhs_len : integer
-- these two tables are created by a parser generator tool

parse_stack : stack of record
    sym : symbol
    st : state

parse_stack.push((null, start_state))
cur_sym : symbol := scan          -- get new token from scanner
loop
    cur_state : state := parse_stack.top.st  -- peek at state at top of stack
    if cur_state = start_state and cur_sym = start_symbol
        return          -- success!
    ar : action_rec := parse_tab[cur_state, cur_sym]
    case ar.action
    shift:
        parse_stack.push((cur_sym, ar.new_state))
        cur_sym := scan          -- get new token from scanner
    reduce:
        cur_sym := prod_tab[ar.prod].lhs
        parse_stack.pop(prod_tab[ar.prod].rhs_len)
    shift_reduce:
        cur_sym := prod_tab[ar.prod].lhs
        parse_stack.pop(prod_tab[ar.prod].rhs_len-1)
    error:
        parse_error

```

Figure 2.28 Driver for a table-driven SLR(1) parser. We call the scanner directly, rather than using the global `input_token` of Figures 2.16 and 2.18, so that we can set `cur_sym` to be an arbitrary symbol.

Top-of-stack state	Current input symbol																		
	<i>sl</i>	<i>s</i>	<i>e</i>	<i>t</i>	<i>f</i>	<i>ao</i>	<i>mo</i>	<i>id</i>	<i>lit</i>	<i>r</i>	<i>w</i>	<i>:=</i>	<i>(</i>	<i>)</i>	<i>+</i>	<i>-</i>	<i>*</i>	<i>/</i>	<i>\$\$</i>
0	s2	b3	-	-	-	-	-	s3	-	s1	s4	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	b5	-	-	-	-	-	-	-	-	-	-	-
2	-	b2	-	-	-	-	-	s3	-	s1	s4	-	-	-	-	-	-	-	b1
3	-	-	-	-	-	-	-	-	-	-	-	s5	-	-	-	-	-	-	-
4	-	-	s6	s7	b9	-	-	b12	b13	-	-	-	s8	-	-	-	-	-	-
5	-	-	s9	s7	b9	-	-	b12	b13	-	-	-	s8	-	-	-	-	-	-
6	-	-	-	-	-	s10	-	r6	-	r6	r6	-	-	-	b14	b15	-	-	r6
7	-	-	-	-	-	-	s11	r7	-	r7	r7	-	-	r7	r7	r7	b16	b17	r7
8	-	-	s12	s7	b9	-	-	b12	b13	-	-	-	s8	-	-	-	-	-	-
9	-	-	-	-	-	s10	-	r4	-	r4	r4	-	-	-	b14	b15	-	-	r4
10	-	-	-	s13	b9	-	-	b12	b13	-	-	-	s8	-	-	-	-	-	-
11	-	-	-	-	b10	-	-	b12	b13	-	-	-	s8	-	-	-	-	-	-
12	-	-	-	-	-	s10	-	-	-	-	-	-	-	b11	b14	b15	-	-	-
13	-	-	-	-	-	-	s11	r8	-	r8	r8	-	-	r8	r8	r8	b16	b17	r8

Figure 2.27 SLR(1) parse table for the calculator language. Table entries indicate whether to shift (s), reduce (r), or shift and then reduce (b). The accompanying number is the new state when shifting, or the production that has been recognized when (shifting and) reducing. Production numbers are given in Figure 2.24. Symbol names have been abbreviated for the sake of formatting. A dash indicates an error. An auxiliary table, not shown here, gives the left-hand-side symbol and right-hand-side length for each production.

table lookup	stack (sym, state) cur_state is the top of the stack	cur_sym	input stream
			read A read ...
	(null,0)	read	A read B...
For each execution of the loop:			
(0, read) → s1	(null, 0)(read, 1)	id(A)	read B sum ...
(1, id) → b5	(null, 0)	stmt	read B sum ...
(0, stmt) → b3	(null, 0)	stmt_list	read B sum ...
(0, stmt_list) → s2	(null, 0)(stmt_list, 2)	read	B sum := ...
(2, read) → s1	(null, 0)(stmt_list, 2)(read, 1)	id(B)	sum := A ...
(1, id) → b5	(null, 0)(stmt_list, 2)	stmt	sum := A ...
(2, stmt) → b2	(null, 0)	stmt_list	sum := A ...
(0, stmt_list) → s2	(null, 0)(stmt_list, 2)	id(sum)	:= A + B ...
(2, id) → s3	(null, 0)(stmt_list, 2)(id, 3)	:=	A + B write ...
(3, :=) → s5	(null, 0)(stmt_list, 2)(id, 3) (:=, 5)	id(A)	+ B write sum ...
(5, id) → b12	(null, 0)(stmt_list, 2)(id, 3) (:=, 5)	factor	+ B write sum ...
(5, factor) → b9	(null, 0)(stmt_list, 2)(id, 3) (:=, 5)	term	+ B write sum ...
(5, term) → s7	(null, 0)(stmt_list, 2)(id, 3) (:=, 5) (term, 7)	+	B write sum ...
(7, +) → r7	(null, 0)(stmt_list, 2)(id, 3) (:=, 5)	expr	+ B write sum ... (reduce puts + back)
(5, expr) → s9	(null, 0)(stmt_list, 2)(id, 3) (:=, 5) (expr, 9)	+	B write sum ...

(9, +)→ b14	(null, 0)(stmt_list, 2)(id, 3) (:=, 5) (expr, 9)	add_op	B write sum ...
(9, add_op)→ s10	(null, 0)(stmt_list, 2)(id, 3) (:=, 5) (expr, 9) (add_op, 10)	id(B)	write sum write ...
(10, id)→ b12	(null, 0)(stmt_list, 2)(id, 3) (:=, 5) (expr, 9) (add_op, 10)	factor	write sum write ...
(10, factor)→ b9	(null, 0)(stmt_list, 2)(id, 3) (:=, 5) (expr, 9) (add_op, 10)	term	write sum write ...
(10, term)→ s13	(null, 0)(stmt_list, 2)(id, 3) (:=, 5) (expr, 9) (add_op, 10) (term, 13)	write	sum write sum ...
(13, write)→ r8	(null, 0)(stmt_list, 2)(id, 3) (:=, 5)	expr	write sum write sum ... (reduce puts write back)
(5, expr)→ s9	(null, 0)(stmt_list, 2)(id, 3) (:=, 5) (expr, 9)	write	sum write sum ...
(9, write)→ r4	(null, 0)(stmt_list, 2)	stmt	write sum write sum ... (reduce puts write back)
(2, stmt)→ b2	(null, 0)	stmt_list	write sum write sum ... (reduce puts write back)
(0, stmt_list)→ s2	(null, 0)(stmt_list, 2)	write	sum write sum ...
(2, write)→ s4	(null, 0)(stmt_list, 2)(write, 4)	id(sum)	write sum / 2 \$\$
(4, id)→ b12	(null, 0)(stmt_list, 2)(write, 4)	factor	write sum / 2 \$\$
(4, factor)→ b9	(null, 0)(stmt_list, 2)(write, 4)	term	write sum / 2 \$\$
(4, term)→ s7	(null, 0)(stmt_list, 2)(write, 4) (term, 7)	write	sum / 2 \$\$
(7, write)→ r7	(null, 0)(stmt_list, 2)(write, 4)	expr	write sum / 2 \$\$ (reduce puts write back)
(4, expr)→ s6	(null, 0)(stmt_list, 2)(write, 4) (expr, 6)	write	sum / 2 \$\$
(6, write)→ r6	(null, 0)(stmt_list, 2)	stmt	write sum / 2 \$\$ (reduce puts write back)
(2, stmt)→ b2	(null, 0)	stmt_list	write sum / 2 \$\$ (reduce puts write back)
(0, stmt_list)→ s2	(null, 0)(stmt_list, 2)	write	sum / 2 \$\$
(2, write)→ s4	(null, 0)(stmt_list, 2) (write, 4)	id(sum)	/ 2 \$\$
(4, id)→ b12	(null, 0)(stmt_list, 2) (write, 4)	factor	/ 2 \$\$
(4, factor)→ b9	(null, 0)(stmt_list, 2) (write, 4)	term	/ 2 \$\$
(4, term)→ s7	(null, 0)(stmt_list, 2) (write, 4) (term, 7)	/	2 \$\$
(7, /)→ b17	(null, 0)(stmt_list, 2) (write, 4) (term, 7)	mult_op	2 \$\$
(7, mult_op)→ s11	(null, 0)(stmt_list, 2) (write, 4) (term, 7)(mult_op, 11)	number(2)	\$\$
(number, 11)→ b13	(null, 0)(stmt_list, 2) (write, 4) (term, 7)(mult_op, 11)	factor	\$\$
(factor, 11)→ b10	(null, 0)(stmt_list, 2) (write, 4)	term	\$\$

(term, 4)→ s7	(null, 0)(stmt_list, 2) (write, 4) (term, 7)	\$\$	
(\$\$, 7)→ r7	(null, 0)(stmt_list, 2) (write, 4)	expr	\$\$ (reduce puts \$\$ back)
(expr, 4)→ s6	(null, 0)(stmt_list, 2) (write, 4) (expr, 6)	\$\$	
(\$\$, 6)→ r6	(null, 0)(stmt_list, 2)	stmt	\$\$ (reduce puts \$\$ back)
(stmt, 2)→ b2	(null, 0)	stmt_list	\$\$
(stmt_list, 0)→ s2	(null, 0)(stmt_list, 2)	\$\$	
(\$\$, 2)→ b1	(null, 0)	program	
cur_state=0 & cur_sym=program so return☺			