

Shift-Reduce Parsing

Prof. James L. Frankel
Harvard University

Version of 6:43 PM 17-Oct-2023
Copyright © 2023, 2022, 2016, 2015 James L. Frankel. All rights reserved.

Bottom-Up Parsing

- Unlike our predictive parsers, a bottom-up parser builds the parse tree from the leaves to the root
- LR(k) parser
 - L for left-to-right scanning of the input
 - R for constructing a rightmost derivation in reverse
 - k for the number of tokens of lookahead required ($k=1$ is the default)
- LR grammars exist for all programming languages
 - More languages can be described by LR than LL grammars
- Efficient implementations are straightforward
- Syntax errors are tagged as soon as possible

LR Grammar

- $(G_1) E \rightarrow E + T$
- $(G_2) E \rightarrow T$
- $(G_3) T \rightarrow T * F$
- $(G_4) T \rightarrow F$
- $(G_5) F \rightarrow (E)$
- $(G_6) F \rightarrow \text{id}$

LR Parsing Table

State	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	S(5)			S(4)			1	2	3
1		S(6)				Accept			
2		R(G ₂)	S(7)		R(G ₂)	R(G ₂)			
3		R(G ₄)	R(G ₄)		R(G ₄)	R(G ₄)			
4	S(5)			S(4)			8	2	3
5		R(G ₆)	R(G ₆)		R(G ₆)	R(G ₆)			
6	S(5)			S(4)				9	3
7	S(5)			S(4)					10
8		S(6)			S(11)				
9		R(G ₁)	S(7)		R(G ₁)	R(G ₁)			
10		R(G ₃)	R(G ₃)		R(G ₃)	R(G ₃)			
11		R(G ₅)	R(G ₅)		R(G ₅)	R(G ₅)			

LR-Parsing Algorithm Initial State

- Start with s_0 on the stack, where s_0 is the initial state
- Start with $\omega\$$ in the input buffer

LR-Parsing Algorithm

```
Let  $a$  be the first symbol of  $\omega\$$ ;  
while(1) {  
  Let  $s$  be the state on top of the stack;  
  if(ACTION[ $s, a$ ] = shift  $t$ ) {  
    push  $t$  onto the stack;  
    Let  $a$  be the next input symbol;  
  } else if(ACTION[ $s, a$ ] = reduce  $A \rightarrow \beta$ ) {  
    pop  $|\beta|$  symbols off the stack;  
    Let state  $t$  now be on top of the stack;  
    push GOTO[ $t, A$ ] onto the stack;  
    output the production  $A \rightarrow \beta$ ;  
  } else if(ACTION[ $s, a$ ] = accept)  
    break;  
  else {  
    call error-recovery routine;  
  }  
}
```

Example of Shift-Reduce Parsing

	Stack	Symbols	Input	Action
1	0		id + id * id \$	Shift 5
2	0 5	id	+ id * id \$	Reduce G_6 ($F \rightarrow id$)
3	0 3	F	+ id * id \$	Reduce G_4 ($T \rightarrow F$)
4	0 2	T	+ id * id \$	Reduce G_2 ($E \rightarrow T$)
5	0 1	E	+ id * id \$	Shift 6
6	0 1 6	E +	id * id \$	Shift 5
7	0 1 6 5	E + id	* id \$	Reduce G_6 ($F \rightarrow id$)
8	0 1 6 3	E + F	* id \$	Reduce G_4 ($T \rightarrow F$)
9	0 1 6 9	E + T	* id \$	Shift 7
10	0 1 6 9 7	E + T *	id \$	Shift 5
11	0 1 6 9 7 5	E + T * id	\$	Reduce G_6 ($F \rightarrow id$)
12	0 1 6 9 7 10	E + T * F	\$	Reduce G_3 ($T \rightarrow T * F$)
13	0 1 6 9	E + T	\$	Reduce G_1 ($E \rightarrow E + T$)
14	0 1	E	\$	Accept