Syntax Analysis:

Context-free Grammars, Pushdown Automata and Parsing Part - 5

Y.N. Srikant

Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Principles of Compiler Design

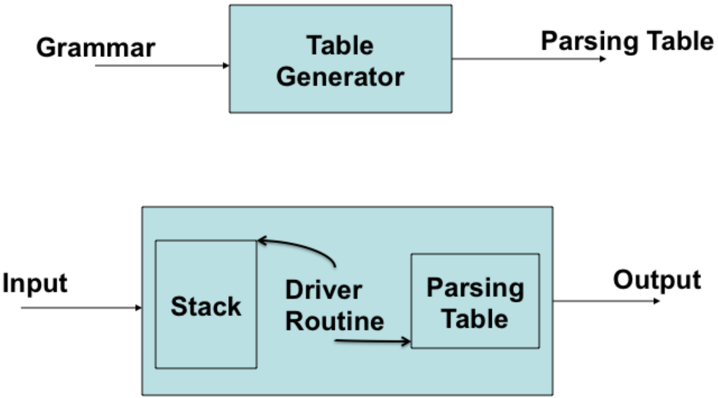
Outline of the Lecture

- What is syntax analysis? (covered in lecture 1)
- Specification of programming languages: context-free grammars (covered in lecture 1)
- Parsing context-free languages: push-down automata (covered in lectures 1 and 2)
- Top-down parsing: LL(1) parsing (covered in lectures 2 and 3)
- Recursive-descent parsing (covered in lecture 4)
- Bottom-up parsing: LR-parsing

LR Parsing

- LR(k) - *L*eft to right scanning with *R*ightmost derivation in reverse, *k* being the number of lookahead tokens
  - $k = 0, 1$ are of practical interest
- LR parsers are also automatically generated using parser generators
- LR grammars are a subset of CFGs for which LR parsers can be constructed
- LR(1) grammars can be written quite easily for practically all programming language constructs for which CFGs can be written
- LR parsing is the most general non-backtracking shift-reduce parsing method (known today)
- LL grammars are a strict subset of LR grammars - an LL(k) grammar is also LR(k), but not vice-versa

LR Parser Generation

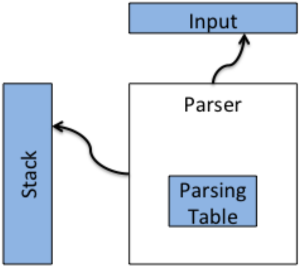


LR Parser Generator

LR Parser Configuration

- A configuration of an LR parser is:
  $(s_0 X_1 s_2 X_2 ... X_m s_m, \quad a_i a_{i+1} ... a_n \$)$, where,
  stack **unexpended input**
  $s_0, s_1, ..., s_m$, are the states of the parser, and $X_1, X_2, ..., X_m$, are grammar symbols (terminals or nonterminals)
- Starting configuration of the parser: $(s_0, a_1 a_2 ... a_n \$)$, where, s_0 is the initial state of the parser, and $a_1 a_2 ... a_n$ is the string to be parsed
- Two parts in the parsing table: *ACTION* and *GOTO*
  - The *ACTION* table can have four types of entries: **shift, reduce, accept**, or **error**
  - The *GOTO* table provides the next state information to be used after a *reduce* move

LR Parsing Algorithm




```
Initial configuration: Stack = state 0, Input = w$,
a = first input symbol;
repeat {
   let s be the top stack state;
   let a be the next input symbol;
   if ( ACTION[s, a] == shift p) {
        push a and p onto the stack (in that order);
        advance input pointer;
   } else if (ACTION[s,a] == reduce A → α) then {
              pop 2*|α| symbols off the stack;
              let s' be the top of stack state now;
              push A and GOTO[s', A] onto the stack
              (in that order);
           } else if (ACTION[s, a] == accept) break;
                 /* parsng is over */
                 else error();
} until true; /* for ever */
```

LR Parsing Example 1 - Parsing Table

STATE	ACTION				GOTO		
	a	b	c	$	S	A	B
0	S2		S3		1		
1				R1 acc			
2	S2	S6	S3		8	4	
3	R3	R3	R3	R3			
4	S2		S3		5		
5	R2	R2	R2	R2			
6	S7						
7	R4	R4	R4	R4			
8	S2	S10	S3		12		9
9	R5	R5	R5	R5			
10	S2	S6	S3		8	11	
11	R6	R6	R6	R6			
12	R7	R7	R7	R7			

1. S' → S
2. S → aAS
3. S → c
4. A → ba
5. A → SB
6. B → bA
7. B → S

LR Parsing Example 1 (contd.)

Stack	**Input**	**Action**
0	$acbbac\$$	S2
$0a2$	$cbbac\$$	S3
$0a2c3$	$bbac\$$	R3 ($S \rightarrow c$, goto(2,S) = 8)
$0a2S8$	$bbac\$$	S10
$0a2S8b10$	$bac\$$	S6
$0a2S8b10b6$	$ac\$$	S7
$0a2S8b10b6a7$	$c\$$	R4 ($A \rightarrow ba$, goto(10,A) = 11)
$0a2S8b10A11$	$c\$$	R6 ($B \rightarrow bA$, goto(8,B) = 9)
$0a2S8B9$	$c\$$	R5 ($A \rightarrow SB$, goto(2,A) = 4)
$0a2A4$	$c\$$	S3
$0a2A4c3$	$\$$	R3 ($S \rightarrow c$, goto(4,S) = 5)
$0a2A4S5$	$\$$	R2 ($S \rightarrow aAS$, goto(0,S) = 1)
$0S1$	$\$$	R1 ($S' \rightarrow S$), and accept

LR Parsing Example 2 - Parsing Table

STATE	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				R7 acc			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

1. E → E+T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → id
7. S → E

LR Parsing Example 2(contd.)

Stack	Input	Action
0	$id + id * id\$$	S5
0 *id* 5	$+id * id\$$	R6 ($F \rightarrow id$, G(0,F) = 3)
0 F 3	$+id * id\$$	R4 ($T \rightarrow F$, G(0,T) = 2)
0 T 2	$+id * id\$$	R2 ($E \rightarrow T$, G(0,E) = 1)
0 E 1	$+id * id\$$	S6
0 E $1 + 6$	$id * id\$$	S5
0 E $1 + 6$ *id* 5	$*id\$$	R6 ($F \rightarrow id$, G(6,F) = 3)
0 E $1 + 6F3$	$*id\$$	R4 ($T \rightarrow F$, G(6,T) = 9)
0 E $1 + 6T9$	$*id\$$	S7
0 E $1 + 6T9 * 7$	$id\$$	S5
0 E $1 + 6T9 * 7$ *id* 5	$\$$	R6 ($F \rightarrow id$, G(7,F) = 10)
0 E $1 + 6T9 * 7F10$	$\$$	R3 ($T \rightarrow T * F$, G(6,T) = 9)
0 E $1 + 6T9$	$\$$	R1 ($E \rightarrow E + T$, G(0,E) = 1)
0 E 1	$\$$	R7 ($S \rightarrow E$) and accept

LR Grammars

- Consider a rightmost derivation:
  $S \Rightarrow^*_{rm} \phi B t \Rightarrow_{rm} \phi\beta t$,
  where the production $B \rightarrow \beta$ has been applied
- A grammar is said to be **LR(k)**, if for any given input string, at each step of any rightmost derivation, the handle β can be detected by examining the string $\phi\beta$ and scanning *at most*, first k symbols of the unused input string t

LR Grammars (contd.)

- Example: The grammar, $\{S \rightarrow E,\ E \rightarrow E + E \mid E * E \mid id\}$, is not LR(2)
  - $S \Rightarrow^1 \underline{E} \Rightarrow^2 \underline{E + E} \Rightarrow^3 E + \underline{E * E} \Rightarrow^4 E + E * \underline{id} \Rightarrow^5 E + \underline{id} * id \Rightarrow^6 \underline{id} + id * id$
  - $S \Rightarrow^{1'} \underline{E} \Rightarrow^{2'} \underline{E * E} \Rightarrow^{3'} E * \underline{id} \Rightarrow^{4'} \underline{E + E} * id \Rightarrow^{5'} E + \underline{id} * id \Rightarrow^{6'} \underline{id} + id * id$
  - In the above two derivations, the handle at steps 6 & 6' and at steps 5 & 5', is $E \rightarrow id$, and the position is underlined (with the same lookahead of two symbols, $id+$ and $+id$)
  - However, the handles at step 4 and at step 4' are different ($E \rightarrow id$ and $E \rightarrow E + E$), even though the lookahead of 2 symbols is the same ($*id$), and the stack is also the same ($\phi = E + E$)
  - That means that the handle cannot be determined using the lookahead

Shift and Reduce Actions

- If a state contains an item of the form $[A \rightarrow \alpha.]$ ("reduce item"), then a reduction by the production $A \rightarrow \alpha$ is the action in that state
- If there are no "reduce items" in a state, then shift is the appropriate action
- There could be shift-reduce conflicts or reduce-reduce conflicts in a state
    - Both shift and reduce items are present in the same state (S-R conflict), or
    - More than one reduce item is present in a state (R-R conflict)
    - It is normal to have more than one shift item in a state (no shift-shift conflicts are possible)
- If there are no S-R or R-R conflicts in any state of an LR(0) DFA, then the grammar is LR(0), otherwise, it is not LR(0)