SLR PARSING (CONTINUED)

I. SUMMARY OF SLR PARSING

The crux of SLR parsing is to determine the steps to be taken for ACTION[*i*,*a*] for the various values of *i* and *a*, where state *i* is constructed from I_i and a is a terminal.

1. If [A → α.a.β] is in I_i and GOTO (I_i , A) = Ij then ACTION [*i* , *a*] is "shift j". This implies that the state j should be pushed on to the state stack and the input pointer must move ahead by one character.

2. If [A → α.] is in I_i and a in FOLLOW (A) then ACTION [*i* , *a*] is "reduce A→α" for all a in FOLLOW (A). A may not be S′, the start state of the augmented LR automaton.
   Here a acts as a watchdog character. If a is in FOLLOW(A), that means that a appears after non-terminal A. This ensures that the reduction gives a valid move of the rightmost derivation.

3. If [S′ → S] is in I_i then ACTION [*i* , $] should be "accept"

ACTION [*i* , *a*] that does not satisfy any of the above is an "error".

Additionally, GOTO (I_i , A) = I_j means that GOTO [i , A] = j

II. DEMONSTRATION OF SLR PARSING

The grammar used is the same as the one used consistently before:

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

The string to be parsed is **id + id * id**
The moves of the LR parser are shown on the next page. For reference:

FOLLOW(E) = { + , $ }
FOLLOW(T) = { * , + , $ }
FOLLOW(F) = { * , + , $ }

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	shift
(2)	0 5	$\mathbf{id}$	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reduce by $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reduce by $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce by $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce by $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce by $E \to E + T$
(14)	0 1	E	$\$$	accept

If the right-most column is now traversed upwards, and the productions by which the reduce steps occur are arranged in that sequence, then that will constitute a leftmost derivation of the string by this grammar. This highlights the bottom-up nature of SLR parsing.

III. THE SLR PARSING TABLE

To understand SLR parsing easily, the SLR parsing table is created using the principles discussed above in Item I.

The rows of this table are represented by the state numbers and the columns are divided into two segments – the ACTION function where the columns are represented by the terminals and the GOTO function where the columns are the non-terminals.

The SLR parsing table for the previous grammar is shown on the next page, while the notation for the table is specified here for easy reference.

1. Shift *i* is represented by "si". (Check the previous notes for the states)
2. Reduction number j (check Item II for numbering) is represented by "rj"
3. Accept is represented by "acc"
4. Error is represented by a blank space in the cell

Using the SLR parsing table, the process of shift-reduce parsing for SLR grammars can be automated and done easily.

STATE	ACTION						GOTO		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				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			

IV. THINGS TO NOTE:

1. CONFLICT FREE NATURE OF SLR

   The above procedure worked for the particular grammar because it is an SLR grammar, i.e one that contains no conflicts. An example of a shift/reduce conflict is shown below:

   A → B.
   A → B. + C

   If the above two items appear in the same state, then there will be a conflict in the case of ACTION [A , +] about whether to shift and move forward in the input or reduce by "A → B".

   Essentially, if each table entry in the parsing table is unique without any duplicity, then there are no conflicts and the grammar in question is SLR.

2. CORRECTNESS OF SLR

   The stack always contains a set of symbols until the input has been matched and only the start state remains - therefore a right sentential derivation can always be found for the given string, if the table is correctly designed without any conflicts. Hence, SLR parsing is correct.