SLR Parser Algorithm

Dr. Meera Thapar Khanna

CSE Department

Pandit Deendayal Energy University

Parsing Algorithm

- A parsing Algorithm uses the current State X, the next input symbol ‘a’ to consult the entry at action[X][a]. it makes one of the four following actions as given below:
- If the action[X][a]=shift Y, the parser executes a shift of Y on to the top of the stack and advances the input pointer.
- If the action[X][a]= reduce Y (Y is the production number reduced in the State X), if the production is Y->β, then the parser pops 2*β symbols from the stack and push Y on to the Stack.
- If the action[X][a]= accept, then the parsing is successful and the input string is accepted.
- If the action[X][a]= error, then the parser has discovered an error and calls the error routine.

Example

Grammar

S -> AA

A -> aA | b

Parsing Table

	a	b	$	S	A
	ACTION			GOTO	
I0	S3	S4		1	2
I1			Accept		
I2	S3	S4			5
I3	S3	S4			6
I4	r3	r3	r3		
I5			r1		
I6	r2	r2	r2		

Stack Implementation for string: ‘aabb’

Stack	Input	Action
0	aabb$	I0 → a, S3
0a3	aabb$	I3→a, S3
0a3a3	aabb$	I3 → b, S4
0a3a3b4	aabb$	I4→b, r3
0a3a3A6	aabb$	I6→b, r2
0a3A6	aabb$	I6→b, r2
0A2	aabb$	I2→b, s4
0A2b4	aabb$	I4→$, r3
0A2A5	aabb$	I5→$,r1
0s1	aabb$	accept

Shift ‘a’ and goto state 3

Reduction means: reduce the previous symbol set to RHS and not reducing the actual symbol at the pointer.

Pop number symbols = Length of RHS * 2

Below ‘A’ is 3, so 3 -> A = i6

Example

Grammar

E -> E + T | T

T -> T * F | F

F -> (E) | id

G:

E -> E + T	(1)
E -> T	(2)
T -> T * F	(3)
T -> F	(4)
F -> (E)	(5)
F -> id	(6)

Parsing Table

	id	+	*	(	)	$	E	T	F
I0	s5						1	2	3
I1		s6				Accept			
I2		r2	s7		r2	r2			
I3		r4	r4		r4	r4			
I4	s5			s4			8	2	3
I5		r6	r6		r6	r6			
I6	s5			S4				9	3
I7	s5			S4					10
I8		s6			S11				
I9		r1	s7		r1	r1			
I10		r3	r3		r3	r3			
I11		r5	r5		r5	r5			

Stack Implementation for string: ‘id*id’

Stack	Input	Action
0	Id*id$	I0 -> id, S5 | GOTO(i0, id) = S5; shift
0id5	*id$	I5 -> *, r6 | GOTO(i5, +) = r6; reduce by F -> id
0F3	*id$	I3 -> *, r4 | GOTO(i0, F) = S; GOTO(i3, +) = r4; reduce by T -> F
0T2	*id$	I2 -> *, S7 | GOTO(i0, T) = S2;
0T2*7	Id$	I7 -> id, S5 |
0T2*7id5	$	I5 -> $, r6
0T2*7F10	$	I10 -> $, r3
0T2	$	I2 -> $, r2
0E1	$	I1 -> $, Accept

Example

Grammar

S-> T*P

T->U

T->T*U

P->Q+P

P->Q

Q->id

U->id

Augmented Grammar

S’->S

S-> T*P

T->U

T->T*U

P->Q+P

P->Q

Q->id

U->id

S' → S —0 ← Add
S → T*P —1
T → U —2
T → T*U —3
P → Q+P —4
P → Q —5
Q → id —6
U → id —7

SLR for given Grammar

I_0
S' → .S
S → .T*P
T → .U
T → .T*U
U → .id

$(I_0, S) \rightarrow I_1$
S' → S.

$(I_0, T) \rightarrow I_2$
S → T.*P
T → T.*U

$(I_0, U) \rightarrow I_3$
T → U.

$(I_0, id) \rightarrow I_4$
U → id.

$(I_2, *) \rightarrow I_5$
S → T*.P
T → T*.U
P → .Q+P
P → .Q
U → .id
Q → .id

$(I_5, P) \rightarrow I_6$
S → T*P.

$(I_5, Q) \rightarrow I_7$
P → Q.+P
P → Q.

$(I_5, U) \rightarrow I_8$
T → T*U.

$(I_5, id) \rightarrow I_9$
U → id.
Q → id.

$(I_7, +) \rightarrow I_{10}$
P → Q+.P
P → .Q
P → .Q+P
Q → .id

$(I_{10}, P) \rightarrow I_{11}$
P → Q+P.

$(I_{10}, Q) \rightarrow I_7$
P → Q.+P
P → Q.

$(I_{10}, id) \rightarrow I_{12}$
Q → id.

First(S) = {id}
First(T) = {id}
First(P) = {id}
First(Q) = {id}
First(U) = {id}

Follow(S) = {$}
Follow(T) = {*}
Follow(P) = {$}
Follow(Q) = {+, $}
Follow(U) = {*}

Parsing Table

States	ACTION					GOTO				
	id	+	*	$	S	T	P	Q	U
I_0	S_4				1	2			3
I_1				accept					
I_2			S5						
I_3			r2						
I_4			r7						
I_5	S9						6	7	8
I_6				r1					
I_7		S10		r5					
I_8			r3						
I_9		r6	r7	r6					
I_{10}	S12						11	7	
I_{11}				r4					
I_{12}		r6		r6					

Stack Implementation for string: ‘id*id’

Stack	Input	Action
0	Id*id$	I0 -> id, S4
0id4	*id$	I4 -> *, r7
0U3	*id$	I3 -> *, r2
0T2	*id$	I2 -> *, S5
0T2*5	Id$	I5 -> id, S9
0T2*5id9	$	I9 -> $, r6
0T2*5Q7	$	I7 -> $, r5
0T2*5P6	$	I6 -> $, r1
0S1	$	I1 -> $, Accept

SLR table conflicts

When multiple entries occur in the table. That is said to be a Conflict.

- Shift-Reduce Conflict in SLR (1) Parsing
- Reduce-Reduce Conflict in SLR (1) Parsing

Shift-Reduce Conflict in SLR (1) Parsing

Shift Reduce Conflict in the LR (1) parsing occurs when a state has :

1. A Reduced item of the form A-> α• and Follow(A) includes the terminal value ‘a’.

2. An incomplete item of the form A-> β•aα as shown below

A-> β•a α (on symbol a going to next state say i_j)

B-> b• (• at the end so reduce production)

Reduce - Reduce Conflict in SLR (1) Parsing

Reduce- Reduce Conflict in the LR (1) parsing occurs when a state has two or more reduced items of the form:

1. A -> α•

2. B -> β• and Follow (A) ∩ Follow(B) ≠ null

Example Grammar:

S-> αAaBa

A-> α

B-> β

Follow(S)= {$} Follow(A)={a} and Follow(B)= {a}

Follow of A and B not null so reduce-reduce conflict.