LAB ASSIGNMENT II

Compiler Construction

LAB ASSIGNMENT II

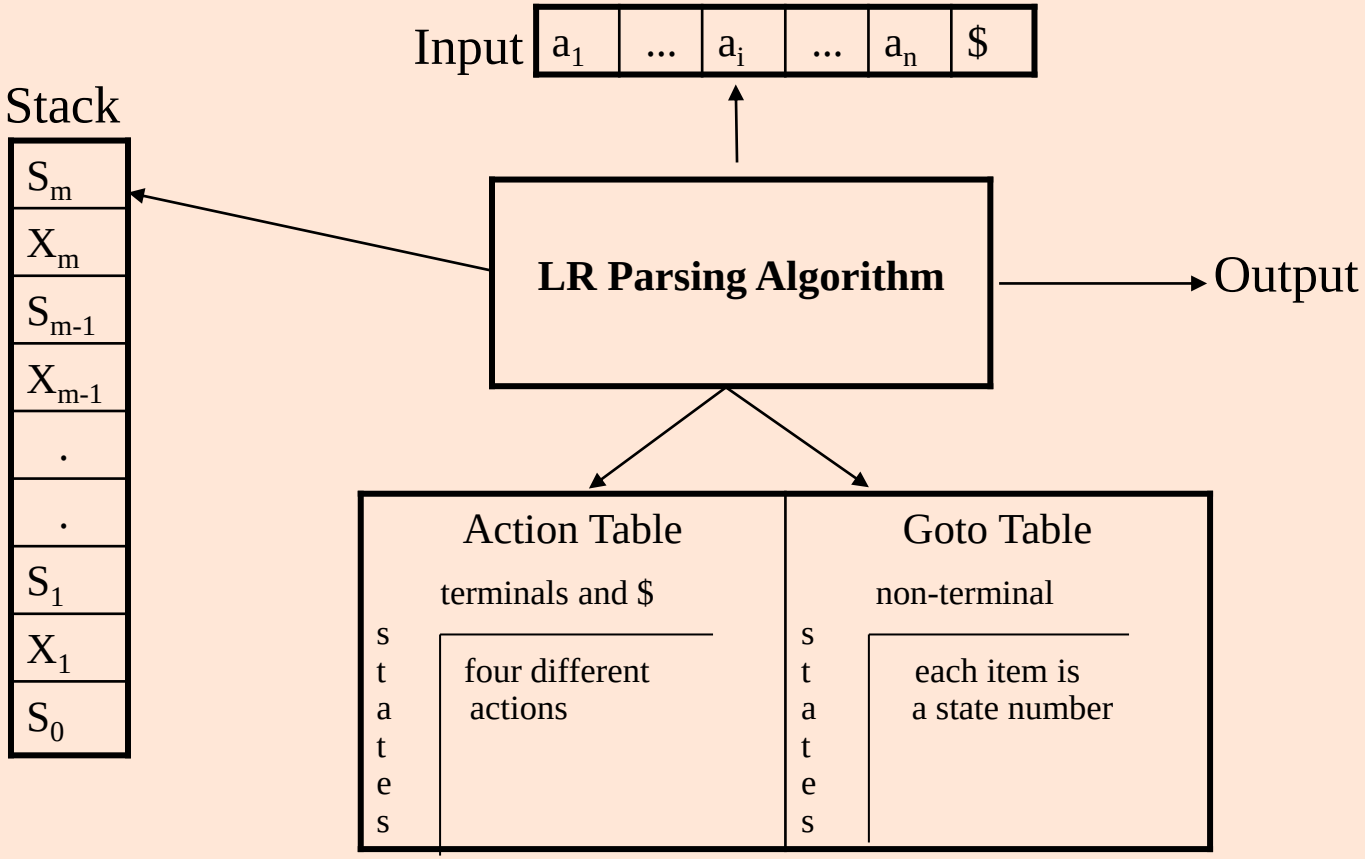
Design a SLR parser for the grammar given below:

E→E+T/T
T→T*F/F
F→(E)/id
This will involve three steps:
Generate the Set of Items (5 Marks) (3 Lab of 2 Hrs.)
Generate the Action and GOTO table (5 marks) (3 Labs of 2 Hrs. each)

OR

Design a SLR parser for the any grammar (Generic)

LR Parsing Algorithm



ACTIONS OF SLR-PARSER

1. **Shift s** -- shifts the next input symbol and the state **s** onto the stack

   $(S_o X_1 S_1 \ldots X_m S_m, a_i a_{i+1} \ldots a_n \$) \rightarrow (S_o X_1 S_1 \ldots X_m S_m a_i S, a_{i+1} \ldots a_n \$)$

2. **Reduce $A \rightarrow \beta$** (or **rn** where n is a production number)
   - pop $2*|\beta|$ (=r) items from the stack; let us assume that $\beta = Y_1 Y_2 \ldots Y_r$
   - then push **A** and **s** where **$s=goto[s_{m-r}, A]$**

   $(S_o X_1 S_1 \ldots X_m S_m, a_i a_{i+1} \ldots a_n \$) \rightarrow (S_o X_1 S_1 \ldots X_{m-r} S_{m-r} A s, a_i \ldots a_n \$)$

   In fact, $Y_1 Y_2 \ldots Y_r$ is a handle.

   $$X_1 \ldots X_{m-r} A a_i \ldots a_n \$ \Rightarrow X_1 \ldots X_m Y_1 \ldots Y_r a_i a_{i+1} \ldots a_n \$$$

   - Output is the reducing production; reduce $A \rightarrow \beta$

3. **Accept** – Parsing successfully completed
4. **Error** -- Parser detected an error (an empty entry in the action table)

Algorithm for Shift Reduce Parser

```
PUSH 0 ON STACK
READ TOKEN
WHILE (0)
{
        STATE = TOP OF STACK
        IF (ACTION[STATE,TOKEN] == 'Si')
        {   PUSH TOKEN AND i ON TO STACK
            READ TOKEN
        }
        ELSE IF(ACTION[STATE,TOKEN] == 'Ri')
        {
                POP 2N items FROM STACK;
                                        /* N is no. of items in right side of production no. i */
                STATE = TOP OF STACK
                                        /*restore state before reduction from top of stack*/
                PUSH GRAMMER-SYMBOL X AND GOTO[STATE,X] ON STACK
                                        /*state after reduction */
        }
        ELSE IF(ACTION[STATE,TOKEN] == 'a'
                SENTENCE PARSED
        ELSE
                ERROR IN SENTENCE
        }
}
```

CONSTRUCTION OF THE CANONICAL SLR COLLECTION

- To create the SLR parsing tables for a grammar G, we will create the canonical LR(0) collection of the grammar G'.

- ***Algorithm***:

  C is { closure({S'→ • S}) }

  repeat the followings until no more set of LR(0) items can be added to ***C***.

  for each I in ***C*** and each grammar symbol X

  if goto(I,X) is not empty and not in ***C***

  add goto(I,X) to ***C***

- goto function is a DFA on the sets in C.

CONSTRUCTION OF SLR ITEM SET FROM THE GIVEN GRAMMAR

closure(I): I is a set of SLR items for a grammar G

put every LR(0) item in I to closure(I)

If $A \rightarrow \alpha.B\beta \in$ closure(I)and $B \rightarrow \gamma$ is a production of G;

add $B \rightarrow .\gamma$ to closure(I)

GOTO(I,X): I is a set of SLR and X is terminal or non-terminal)

If $A \rightarrow \alpha.X\beta$ is SLR item in I

Add all SLR items in **closure({$A \rightarrow \alpha X.\beta$})** in goto(I,X).

The Canonical SLR Collection

I_0
- **E' → •E**
- E → •E+T
- E → •T
- T → •T*F
- T → •F
- F → •(E)
- F → •id

I_1
- E' → E• √
- E → E•+T

I_2
- E → T• √
- T → T•*F

I_3
- T → F• √

I_4
- F → (•**E**)
- E → •E+T
- E → •T
- T → •T*F
- T → •F
- F → • (E)
- F → •id

I_5
- F → id• √

I_6
- E → E+•**T**
- T → •T*F
- T → •F
- F → • (E)
- F → •id

I_7
- T → T*•**F**
- F → •(E)
- F → •id

I_8
- F → (E•)
- E → E•+T

I_9
- E → E+T• √
- T → T•*F

I_{10}
- T → T*F• √

I_{11}
- F → (E) • √

Initial Set
- E′ → •E
- 1) E → •E+T
- 2) E → •T
- 3) T → •T*F
- 4) T → •F
- 5) F → • (E)
- 6) F → •id

CONSTRUCTING THE PARSING TABLE

ACTION

If a is a terminal

$A \rightarrow \alpha.a\beta$ in I_i and goto(I_i,a)=I_j

then action[i,a] is *shift j*.

If $A \rightarrow \alpha.$ is in I_i

then action[i,a] is *reduce* $A \rightarrow \alpha$ for all a in FOLLOW(A) where $A \neq S'$.

If $S' \rightarrow S.$ is in I_i

then action[i,$] is *accept*.

If any conflicting actions generated by these rules

then the grammar is not SLR(1).

GOTO

for all non-terminals A

if goto(I_i,A)=I_j

then goto[i,A]=j

First and Follow Set

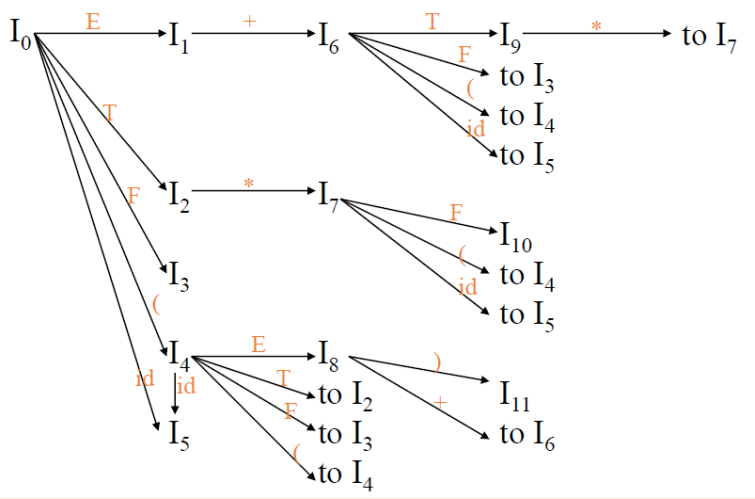
Initial Set
E′ → •E
1) E → •E+T
2) E → •T
3) T → •T*F
4) T → •F
5) F → • (E)
6) F → •id

FIRST(E) = {(,id}
FIRST (T)= {(,id}
FIRST (F)= {(,id}

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

ACTION AND GOTO TABLE

E' → .E
1) E → E+T
2) E → T
3) T → T*F
4) T → F
5) F → (E)
6) F → id



	ACTION							GOTO		
state	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				

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

- S*i* means shift and stack state *i*
- r*j* means reduce by production numbered *j*
- *acc* means accept state
- blank mean error

Desired Output

Input will be any RE and output should be ‘Accept’ if RE is parsed successfully and it should be ‘Reject’ if RE is not parsed.

TOTAL MARKS: 10