Assignment 2: CS 335 (Spring 2020)

I pledge on my honor that I have not given or received any unauthorized assistance.

Name: Deepesh Kumar Lall   Roll No.: 170236

1. **Solution:** As given in the question our grammar is:

$$S \rightarrow (L) \mid a$$
$$L \rightarrow L , S \mid L S \mid b$$

As the above grammar is left recursive Hence currently can't be used to make a predictive parser.

Algorithm 1: Eliminate Left Recursion

Result: Grammar with no left recursion.
Assign an ordering $A_1, \ldots, A_n$ to the nonterminals of the grammar.
for i := 1 to n do begin
  for j:=1 to i-1 do begin
    for each production of the form $A_i \rightarrow A_j\alpha$ do begin
      remove $A_i \rightarrow A_j\alpha$ from the grammar
      for each production of the form $A_j \rightarrow \beta$ do begin
        add $A_i \rightarrow \beta\alpha$ to the grammar
      end
    end
  end
  transform the A_i-productions to eliminate direct left recursion
end

Applying the above algorithm for our grammer we get:

$$S \rightarrow (L) \mid a$$
$$L \rightarrow b\ L_1$$
$$L_1 \rightarrow , SL_1 \mid SL_1 \mid \epsilon$$

Now as this grammar is already left-factored so, Now we can make a predictive parser using this grammar. Thereby finding the FIRST for each nonterminals of grammar.

- FIRST(S) = { (, a }
- FIRST(L_1) = { (, a , ϵ }
- FIRST(L) = { b }

Similarly finding the FOLLOW for each nonterminal of grammar.

- FOLLOW(S) = { $, (, a , , }
- FOLLOW(L_1) = {) }
- FOLLOW(L) = {) }

Algorithm 2: Construction of a predictive parsing table.

Result: Parsing table M.
INPUT: Grammar G .
For each production A $\rightarrow \alpha$ of the grammar, do the following:
  1. For each terminal a in FIRST (α), add A $\rightarrow \alpha$ to M [A, a].
  2. If ϵ is in FIRST (α), then for each terminal b in FOLLOW (A), add A $\rightarrow \alpha$ to M [A, b]. If ϵ is in FIRST (α) and $ is in FOLLOW (A), add A $\rightarrow \alpha$ to M [A, $] as well.
If, after performing the above, there is no production at all in M [A, a], then set M [A, a] to error (which we normally represent by an empty entry in the table).

NON TERMINAL	INPUT SYMBOLS					
	a	b	(	,	)	$
S	S $\rightarrow$ a		S $\rightarrow$ (L)			
L		L $\rightarrow$ bL_1				
L_1	$L_1 \rightarrow$ SL_1		$L_1 \rightarrow$SL_1	$L_1 \rightarrow$,SL_1	$L_1 \rightarrow \epsilon$	

[] Predictive Parsing Table for the Left factorized and factored grammar according to algorithm 2.

□

2. **Solution:** As given in the question our grammar is:

$$S \rightarrow Lp \mid qLr \mid sr \mid qsp$$
$$L \rightarrow s$$

Now considering the above grammar for:

SLR:

Augmenting the above Grammar gives us:

$$S' \rightarrow S$$
$$S \rightarrow Lp \mid qLr \mid sr \mid qsp$$
$$L \rightarrow s$$

Consdering the below algorithm for Computing Cannonical collection of sets of SLR items.

Algorithm 3: Computing cannonical structure of sets of LR(0) items.

Result: All Cannonical structure of sets of LR(0) items
INPUT: Augmented Grammar G .
C = { CLOSURE ({ [S' *to* .S] }) } ;
repeat
  for (each set of items I in C)
    for (each grammar symbol X)
      if (GOTO (I, X) is not empty and not in C)
        add GOTO (I, X) to C ;
until no new sets of items are added to C on a round;

Using the above algorithm we can make cannonical LR(0) items for SLR parsing table construction. Lets say the item are as I_0, I_1..

I_0 : $S' \rightarrow .S$; $S \rightarrow .Lp|.qLr|.sr|.qsp$; $L \rightarrow .s$

I_1 : $S' \rightarrow S.$

I_2 : $S \rightarrow q.Lr|q.sp$; $L \rightarrow .s$

I_3 : $S \rightarrow L.p$

I_4 : $S \rightarrow S.r$; $L \rightarrow s.$

I_5 : $S \rightarrow Lp.$

I_6 : $S \rightarrow sr.$

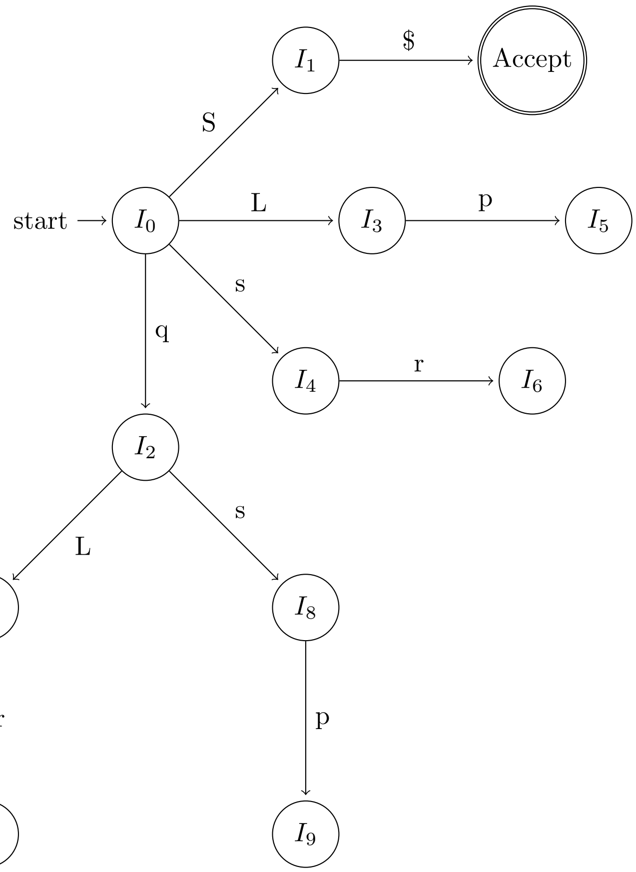
I_7 : $S \rightarrow qL.r$

I_8 : $S \rightarrow qs.p$; $L \rightarrow s.$

I_9 : $S \rightarrow qsp.$

I_{10} : $S \rightarrow qLr.$

Considering all of the above cannonical structure we can have a DFA as below, with all the edges representing the terminal or Nonterminal lookahead value and transition to be taken from 1 state to another when we find it(terminal or non terminal) on top of stack.
The accept state is being reached upon scanning $ from the input and successfull reduction.



Considering the above state transition diagram we need to find the FIRST of all the non terminals:

- FIRST(S) = { s , q }
- FIRST(L) = { s }

Similarly now finding the FOLLOW of all non terminals:

- FOLLOW(S) = { \$ }
- FOLLOW(L) = { p , r }

Defining all the reductions as :

1. S → Lp
2. S → qLr
3. S → sr
4. S → qsp
5. L → s

,and using them as r1,r2..etc with ri implying reduction by ith rule and shift operation as s1,s2...etc with si implying shift to ith state.
We can now form The SLR parse table according to the below algorithm as:

Algorithm 4: Constructing an SLR-parsing table.

SLR-parsing table functions ACTION and GOTO for G'. INPUT: Augmented Grammar G' .

(a) Construct C = { $I_0, I_1..I_{10}$ }, the collection of sets of LR(0) items for G' .

(b) State i is constructed from I_i . The parsing actions for state i are deter- mined as follows:

   i. If [A $\rightarrow \alpha$.aβ] is in I_i and GOTO (I_i , a) = I_j , then set ACTION [i, a] to "shift j ." Here a must be a terminal.

   ii. If [A $\rightarrow \alpha$.] is in I_i , then set ACTION [i, a] to "reduce A $\rightarrow \alpha$ " for all a in FOLLOW (A); here A may not be S' .

   iii. If [$S' \rightarrow S$.] is in I_i , then set ACTION [i, $] to "accept."

   If any conficting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.

(c) The goto transitions for state i are constructed for all nonterminals A using the rule: If GOTO (I_i , A) = I_j , then GOTO [i, A] = j .

(d) All entries not defined by rules (2) and (3) are made "error."

(e) The initial state of the parser is the one constructed from the set of items containing [$S' \rightarrow .S$].

Thereby following the above algorithm we get the Parsing table as :

STATE	ACTION					GOTO	
	p	q	r	s	$	S	L
0		s2		s4		s1	3
1					accept		
2				s8			7
3	s5						
4	r5		s6,r5				
5					r1		
6					r3		
7			s10				
8	s9,r5		r5				
9					r4		
10					r2		

Thereby, this gives us a **shift reduce conflict** as highlighted in red Hence its not SLR(1).

LALR:

Augmenting the above Grammar gives us:

$$S' \rightarrow \text{S}$$
$$\text{S} \rightarrow \text{Lp} \mid \text{qLr} \mid \text{sr} \mid \text{qsp}$$
$$\text{L} \rightarrow \text{s}$$

Consdering the below algorithm for Computing Cannonical collection of sets of LALR items.

Algorithm 5: CLOSURE for LR(1) item

Result: CLOSURE for State
INPUT: Item I
repeat
  for (each item [A $\rightarrow \alpha.B\beta$, a] in I)
    for (each production B $\rightarrow \gamma$ in G')
      for (each terminal b in FIRST (βa))   add [B $\rightarrow .\gamma$, b] to set I ; until no more items
 are added to I ;
return I;

Algorithm 6: GOTO for LR(1) item

Result: Return state of GOTO with a lookahead propogated and spontaneous.
INPUT: Item I and Non Terminal X
initialize J to be the empty set;
for (each item [A $\rightarrow \alpha.X\beta$, a] in I)
  add item [A $\rightarrow \alpha X.\beta$, a] to set J ;
return CLOSURE (J);

Algorithm 7: Computing cannonical structure of sets of LR(1) items.

Result: All Cannonical structure of sets of LR(1) items
INPUT: Augmented Grammar G .
C = { CLOSURE ({ [S' *to* .S , $] }) } ;
repeat
  for (each set of items I in C)
    for (each grammar symbol X)
      if (GOTO (I, X) is not empty and not in C)
        add GOTO (I, X) to C ;
until no new sets of items are added to C on a round;

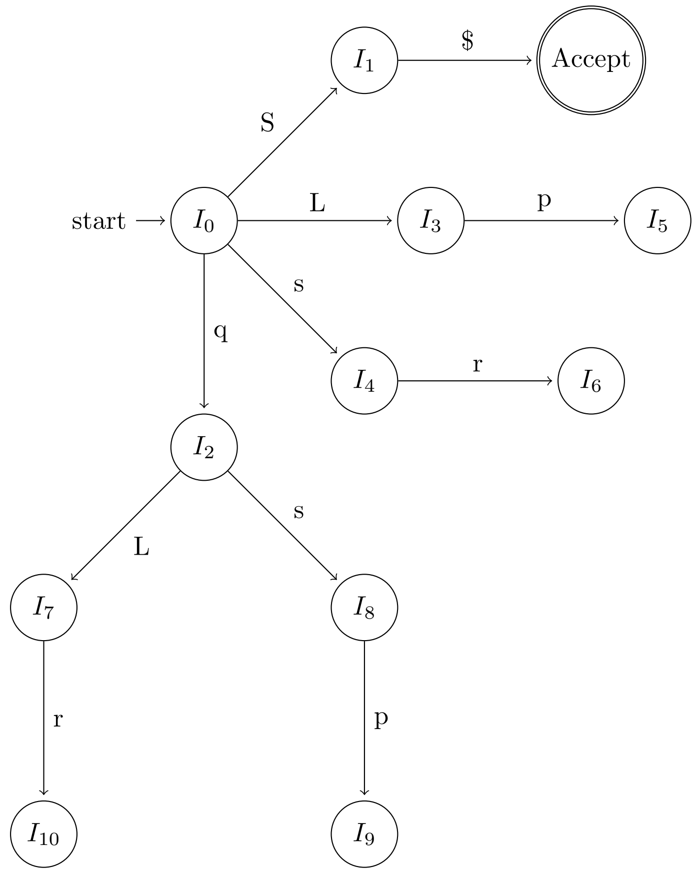
Using the above algorithm we can make cannonical LR(1) items for SLR parsing table construction. Lets say the item are as I_0, I_1..

$\boxed{\begin{array}{l} I_0: \\ \quad S' \rightarrow .S, \$ \\ \quad S \rightarrow .Lp|.qLr|.sr|.qsp, \$ \\ \quad L \rightarrow .s, p \end{array}}$ $\boxed{\begin{array}{l} I_1: \\ \quad S' \rightarrow S., \$ \end{array}}$. $\boxed{\begin{array}{l} I_2: \\ \quad S \rightarrow q.Lr|q.sp, \$ \\ \quad L \rightarrow .s, \{r, s\} \end{array}}$ $\boxed{\begin{array}{l} I_3: \\ \quad S \rightarrow L.p, \$ \end{array}}$. $\boxed{\begin{array}{l} I_4: \\ \quad S \rightarrow S.r, \$ \\ \quad L \rightarrow s., p \end{array}}$ $\boxed{\begin{array}{l} I_5: \\ \quad S \rightarrow Lp., \$ \end{array}}$.

$\boxed{\begin{array}{l} I_6: \\ \quad S \rightarrow sr., \$ \end{array}}$. $\boxed{\begin{array}{l} I_7: \\ \quad S \rightarrow qL.r, \$ \end{array}}$. $\boxed{\begin{array}{l} I_8: \\ \quad S \rightarrow qs.p, \$ \\ \quad L \rightarrow s, \{r, s\}. \end{array}}$. $\boxed{\begin{array}{l} I_9: \\ \quad S \rightarrow qsp., \$ \end{array}}$. $\boxed{\begin{array}{l} I_{10}: \\ \quad S \rightarrow qLr., \$ \end{array}}$.

Considering all of the above cannonical structure we can have a DFA as below, with all the edges representing the terminal or Nonterminal lookahead value and transition to be taken from 1 state to another when we find it(terminal or non terminal) on top of stack.

The accept state is being reached upon scanning $ from the input and successfull reduction.
As The grammar body didnt change for any item and only lookahead changed so there's no change in the DFA.



We dont need to see the FOLLOW as in LALR(1) our lookahead is already been considered in the state diagram.

Defining all the reductions as :

1. S $\rightarrow$ Lp
2. S $\rightarrow$ qLr
3. S $\rightarrow$ sr
4. S $\rightarrow$ qsp
5. L $\rightarrow$ s

,and using them as r1,r2..etc with ri implying reduction by ith rule and shift operation as s1,s2...etc with si implying shift to ith state.

We can now form The LALR parse table according to the below algorithm as:

Algorithm 8: Constructing an LALR-parsing table.

LALR-parsing table functions ACTION and GOTO for G'. INPUT: Augmented Grammar G' .

(a) Construct C = { $I_0, I_1..I_{10}$ }, the collection of sets of LR(1) items for G' .

(b) For each core present among the set of LR(1) items, find all sets having that core, and replace these sets by their union.

(c) Let $C' = \{J_0, J_1, ..., J_m\}$ be the resulting sets of LR(1) items. The parsing actions for state i are constructed from J_i in the same manner. If there is a parsing action conflict, the algorithm fails to produce a parser, and the grammar is said not to be LALR(1).

(d) The GOTO table is constructed as follows. If J is the union of one or more sets of LR(1) items, that is, J = $I_1 \bigcup I_2 \bigcup ... \bigcup I_k$, then the cores of GOTO (I_1 , X), GOTO (I_2 , X) , ... , GOTO (I_k , X) are the same, since I_1 ; I_2 , ,,, , I_k all have the same core. Let K be the union of all sets of items having the same core as GOTO (I_1 ; X). Then GOTO (J, X) = K .

Thereby following the above algorithm we get the Parsing table as :

STATE	ACTION					GOTO	
	p	q	r	s	$	S	L
0		s2		s4		s1	3
1					accept		
2				s8			7
3	s5						
4	r5		s6				
5					r1		
6					r3		
7			s10				
8	s9		r5				
9					r4		
10					r2		

Thereby, this gives us LALR(1) parsing table.

□

3. **Solution:**

As we did in previous part for SLR parser generator similarly Our grammar this time:

$$R \rightarrow R\ '|'\ R$$
$$R \rightarrow RR$$
$$R \rightarrow R*$$
$$R \rightarrow (R)$$
$$R \rightarrow a \mid b$$

Thereby making LR(0) cannonical item using augmented grammer as starting state:(Using Algorithm 3)

I_0 :
- $R' \rightarrow .R$
- R → .R '|' R
- R → .RR
- R → .R*
- R → .(R)
- R → .a
- R → .b

I_1 :
- $R' \rightarrow R.$
- R → R. '|' R
- R → R.R
- R → R.*
- R → .R '|' R
- R → .RR
- R → .R*
- R → .(R)
- R → .a
- R → .b

I_4 :
- R → (.R)
- R → .R '|' R
- R → .RR
- R → .R*
- R → .(R)
- R → .a
- R → .b

I_5 :
- R → R '|'. R
- R → .R '|' R
- R → .RR
- R → .R*
- R → .(R)
- R → .a
- R → .b

I_6 :
- R → (R.)
- R → R. '|' R
- R → R.R
- R → R.*
- R → .R '|' R
- R → .RR
- R → .R*
- R → .(R)
- R → .a
- R → .b

I_7 :
- R → RR.
- R → R. '|' R
- R → R.R
- R → R.*
- R → .R '|' R
- R → .RR
- R → .R*
- R → .(R)
- R → .a
- R → .b

I_2 :
- R → a.

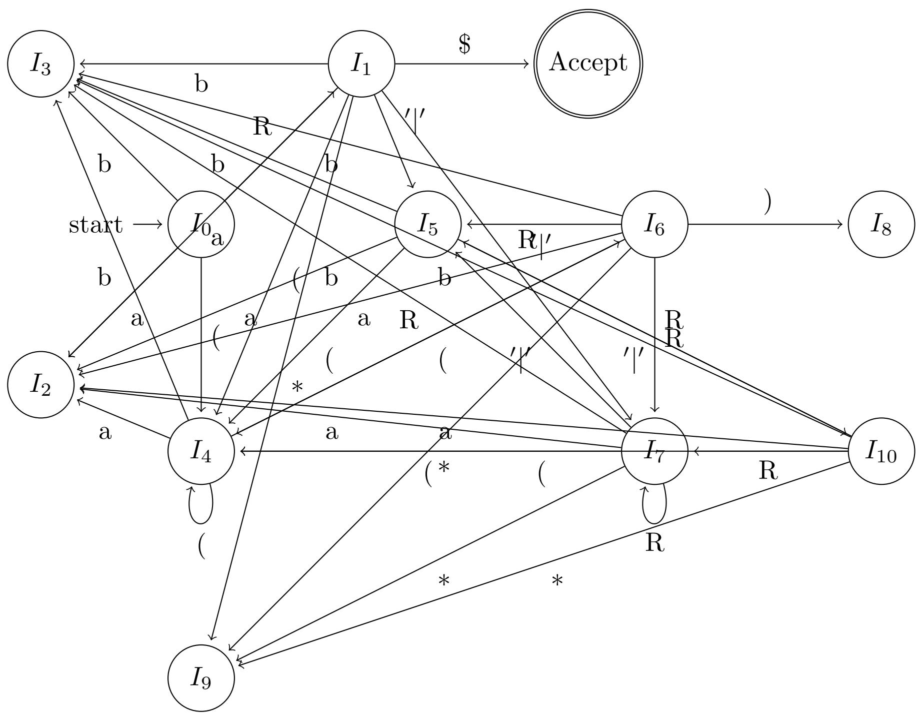
I_3 :
- R → b.

I_8 :
- R → (R).

I_9 :
- R → R*.

I_{10} :
- R → R '|' R.
- R → R. '|' R
- R → R.R
- R → R.*
- R → .R '|' R
- R → .RR
- R → .R*
- R → .(R)
- R → .a
- R → .b

Considering all of the above cannonical structure we can have a DFA as below, with all the edges representing the terminal or Nonterminal lookahead value and transition to be taken from 1 state to another when we find it(terminal or non terminal) on top of stack.
The accept state is being reached upon scanning $ from the input and successfull reduction.



Now knowing the DFA we can find the FIRST and FOLLOW of all non Terminal i.e. R as:

- FIRST(R) = { a , b , (}

- FOLLOW(R) = { \$, $'|'$, $*$,) , a , b , (}

And Now numbering all the productions:

1. R $\rightarrow$ R $'|'$ R
2. R $\rightarrow$ RR
3. R $\rightarrow$ R*
4. R $\rightarrow$ (R)
5. R $\rightarrow$ a | b

Using the above DFA and FOLLOW() of Non terminal we can make the SLR parse table according to Algorithm 4 as done previously using ri and si as entries for i in 1 to n:

[]

STATE	ACTION							GOTO
	a	b	(	*	)	$'|'$	\$	R
0	s2	s3	s4					1
1	s2	s3	s4	s9		s5	accept	7
2	r5	r5	r5	r5	r5	r5	r5	
3	r6	r6	r6	r6	r6	r6	r6	
4	s2	s3	s4					6
5	s2	s3	s4					10
6	s2	s3	s4	s9	s8	s5		7
7	s2,r2	s3,r2	s4,r2	s9,r2	r2	s5,r2	r2	7
8	r4	r4	r4	r4	r4	r4	r4	
9	r3	r3	r3	r3	r3	r3	r3	
10	s2,r1	s3,r1	s4,r1	s9,r1	r1	s5,r1	r1	7

Since there's shift-reduce conflict so It cant be currently used for SLR parsing.
As all the Shift Reduce conflict are due to the Ambiguity arrised from lack of precedence and Associativity of The Production. There by We can remove the Ambiguity by adding Precedence and Associativity to In production 1 and 2. As in every production precedence of operator at lower level dominate over the Production on higher level in recursion Stack (i.e. close to R) Hence Shift is being favored every time before reduction. There by, disambiguous grammar have Shift Rule in all place of Shift Reduce Conflict in ACTION of ambiguose Grammar and the parsing table now is as below.

[]

STATE	ACTION							GOTO
	a	b	(	*	)	$'|'$	\$	R
0	s2	s3	s4					1
1	s2	s3	s4	s9		s5	accept	7
2	r5	r5	r5	r5	r5	r5	r5	
3	r6	r6	r6	r6	r6	r6	r6	
4	s2	s3	s4					6
5	s2	s3	s4					10
6	s2	s3	s4	s9	s8	s5		7
7	s2	s3	s4	s9	r2	s5	r2	7
8	r4	r4	r4	r4	r4	r4	r4	
9	r3	r3	r3	r3	r3	r3	r3	
10	s2	s3	s4	s9	r1	s5	r1	7

□

4. **Solution:**
   This part of assignment is on Parsing a dissertation consists of a title and one or more chapters.The Solution folder Consist of a parser File named **parser.y** and a Lexical Analyzer named **lexer.l**.The Folder also contain a Script for running the commands necessary and is named **runme.sh**.

Running Script:

chmod u+x runme.sh
./runme.sh parser.y lexer.l <Test_File_Name>

Without Running Script:

yacc -dv parser.y
lex lexer.l
gcc y.tab.c lex.yy.c -o output
./output <Test_File_Name> "output.txt" 2¿"error.txt"
(Removinf useless file at end of execution)
rm -rf lex.yy.c y.tab.c y.tab.h y.output

The Output of the above command will be in a file named **output.txt** and all the error messages in **error.txt**.
□