Assignment 2: CS 335

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

Name: Snehal Raj Roll No.: 170705

1. As we see that the grammar contains left recursion, we create an equivalent grammar to avoid the same. To start, we introduce a new non-terminal namely L' and then create the following grammar

   $S \rightarrow (L)|a$
   $L \rightarrow bL'$
   $L' \rightarrow, SL'|SL'|\epsilon$

   We can see that the above grammar is free from left recursion so we proceed to create the parse table.

   To begin, we create the Fist and Follow sets of all three non-terminals which are listed below,

   $First(S) = \{c, a\}$
   $First(L) = \{b\}$
   $First(L') = \{ , c, a, \epsilon\}$

   $Follow(S) = \{\$\}$
   $Follow(L) = \{)\}$
   $Follow(L') = \{)\}$

   So, having constructed the First and the Follow sets, we proceed to build the parsing table which is as shown below

	(	)	a	b		$
S	$S \rightarrow (L)$		$S \rightarrow a$			
L				$L \rightarrow b$		
L'	$L' \rightarrow SL'$	$L \rightarrow \epsilon$	$L' \rightarrow SL'$		$L' \rightarrow, SL'$	

2. In addition to the rules mentioned, we add a dummy rule $S' \rightarrow S$ to augment the grammar. Now, following the procedure for constructing LR(1) parser, the state transitions are as follows,
   $I_0 : (0)S' \rightarrow .S, \$$
   $(1)S \rightarrow .Lp, \$$
   $(2)S \rightarrow .qLr, \$$
   $(3)S \rightarrow .sr, \$$
   $(4)S \rightarrow .qsp, \$$
   $(5)S \rightarrow .s, p$

   $goto(I_0, S) = I_1 :$
   $S' \rightarrow S., \$$

   $goto(I_0, L) = I_2 :$
   $S \rightarrow L.p, \$$

$goto(I_0, q) = I_3 :$
$S \rightarrow q.Lr, \$$
$S \rightarrow q.sp, \$$
$L \rightarrow .s, r$

$goto(I_0, s) = I_4 :$
$S \rightarrow s.r, \$$
$L \rightarrow s., p$

$goto(I_2, p) = I_5 :$
$S \rightarrow Lp., \$$

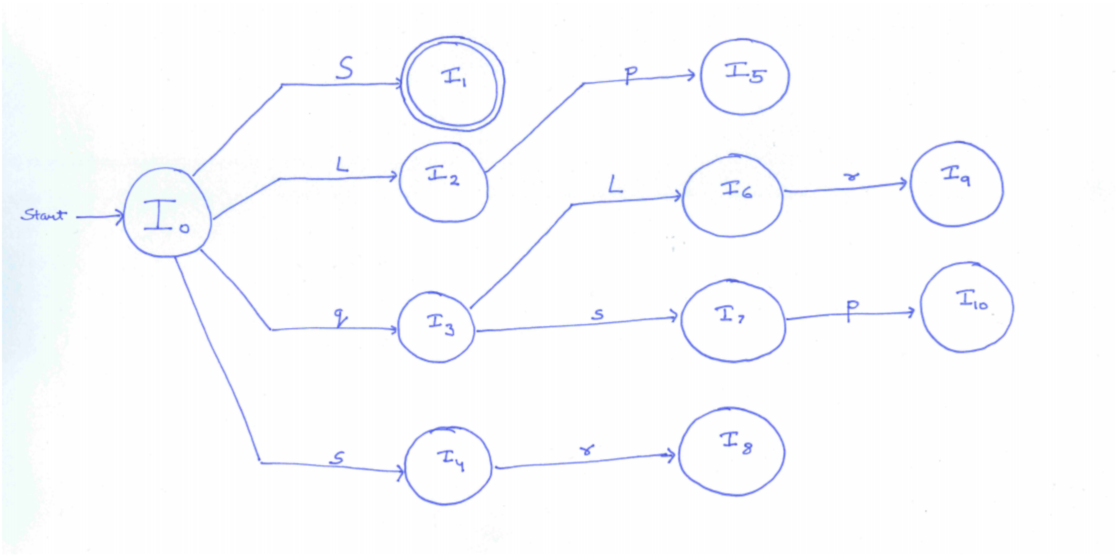
$goto(I_3, L) = I_6 :$
$S \rightarrow qL.r, \$$

$goto(I_3, s) = I_7 :$
$S \rightarrow qs.p, \$$
$L \rightarrow s., r$

$goto(I_4, r) = I_8 :$
$S \rightarrow sr., \$$

$goto(I_6, r) = I_9 :$
$S \rightarrow qLr., \$$

$goto(I_7, p) = I_{10} :$
$S \rightarrow qsp., \$$



As, there are no conflicts and no common first components, the grammar is indeed LALR(1). The parsing table is as follows:

State	Action	Action	Action	Action	Action	Action	Action
	p	q	r	s	\$	S	L
0		s3		s4		1	2
1					accept		
2	s5						
3				s7			6
4	r5		s8				
5					r1		
6			s9				
7	s10		r5				
8					r3		
9					r2		
10					r4		

Now, we follow similar procedures but do not include lookahead in items to obtain the state diagram from SLR(1) parser. The state transitions are as mentioned below

$I_0 : (0)S' \rightarrow .S$

$(1)S \rightarrow .Lp$

$(2)S \rightarrow .qLr$

$(3)S \rightarrow .sr$

$(4)S \rightarrow .qsp$

$(5)S \rightarrow .s$

$goto(I_0, S) = I_1 :$

$S' \rightarrow S.$

$goto(I_0, L) = I_2 :$

$S \rightarrow L.p$

$goto(I_0, q) = I_3 :$

$S \to q.Lr$
$S \to q.sp$
$L \to .s$

$goto(I_0, s) = I_4$:
$S \to s.r$
$L \to s.$

$goto(I_2, p) = I_5$:
$S \to Lp.$

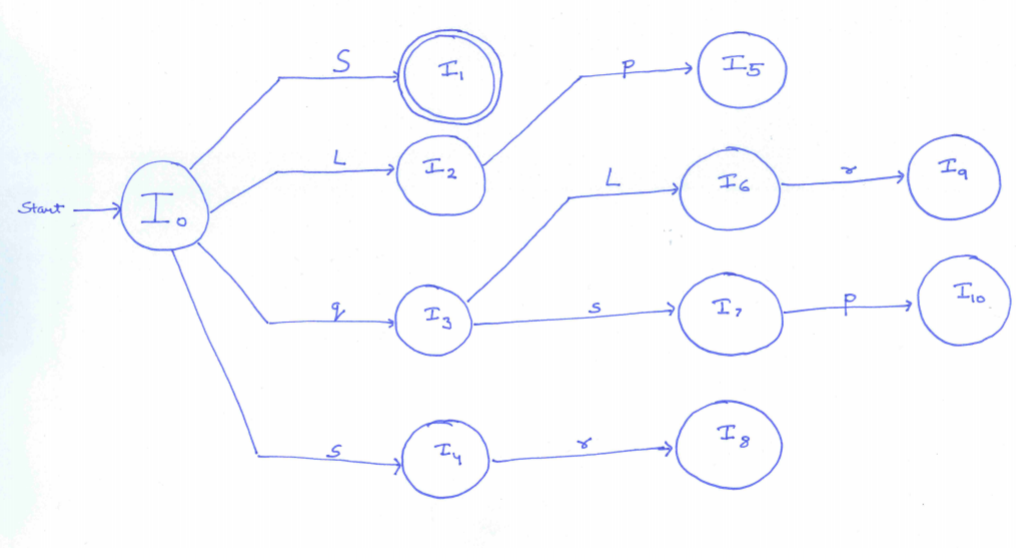
$goto(I_3, L) = I_6$:
$S \to qL.r$

$goto(I_3, s) = I_7$:
$S \to qs.p$
$L \to s.$

$goto(I_4, r) = I_8$:
$S \to sr.$

$goto(I_6, r) = I_9$:
$S \to qLr.$

$goto(I_7, p) = I_{10}$:
$S \to qsp.$



Now for this to be a SLR(1) grammar, there musn't be any conflicts while generating the parsing table. However, as we know that Follow(A)={a,c}, wee that in states I_4 and I_7, there exist shift-reduce conflicts. Hence, the given grammar is not in SLR(1).

3. In addition to the rules mentioned, we add a dummy rule $R' \rightarrow R$ to augment the grammar. Now, following the state transition rules for the parser, the state transitions are as follows,

$I_0 : (0)R' \rightarrow .R$
$(1)R \rightarrow .R|R$
$(2)R \rightarrow .RR$
$(3)R \rightarrow .R*$
$(4)R \rightarrow .(R)$
$(5)R \rightarrow .a$
$(6)R \rightarrow .b$

$goto(I_0, R) = I_1 :$
$R \rightarrow R.$
$R \rightarrow R.|R$
$R \rightarrow R.R$
$R \rightarrow R.*$
$R \rightarrow .R|R$
$R \rightarrow .RR$
$R \rightarrow .R*$
$R \rightarrow .(R)$
$R \rightarrow .a$
$R \rightarrow .b$

$goto(I_0, () = I_2 :$
$R \rightarrow (.R)$
$R \rightarrow .R|R$
$R \rightarrow .RR$
$R \rightarrow .R*$
$R \rightarrow .(R)$
$R \rightarrow .a$
$R \rightarrow .b$

$goto(I_0, a) = I_3 :$
$R \rightarrow a.$

$goto(I_0, b) = I_4 :$
$R \rightarrow b.$

$goto(I_1, |) = I_5 :$
$R \rightarrow R|.R$
$R \rightarrow .R|R$
$R \rightarrow .RR$
$R \rightarrow .R*$

$R \to .(R)$
$R \to .a$
$R \to .b$

$goto(I_1, R) = I_6 :$
$R \to RR.$
$R \to R.|R$
$R \to R.R$
$R \to R.*$
$R \to .R|R$
$R \to .RR$
$R \to .R*$
$R \to .(R)$
$R \to .a$
$R \to .b$
$R \to R * .$

$goto(I_1, *) = I_7 :$
$R \to R * .$

$goto(I_1, () = I_2$
$goto(I_1, a) = I_3$
$goto(I_1, b) = I_4$

$goto(I_2, R) = I_8 :$
$R \to (R.)$
$R \to R.|R$
$R \to R.R$
$R \to R.*$
$R \to .R|R$
$R \to .RR$
$R \to .R*$
$R \to .(R)$
$R \to .a$
$R \to .b$

$goto(I_2, () = I_2$
$goto(I_2, a) = I_3$
$goto(I_2, b) = I_4$

$goto(I_5, R) = I_9$
$R \to R|R.$
$R \to R.R$
$R \to R.R$
$R \to R.*$
$R \to .R|R$

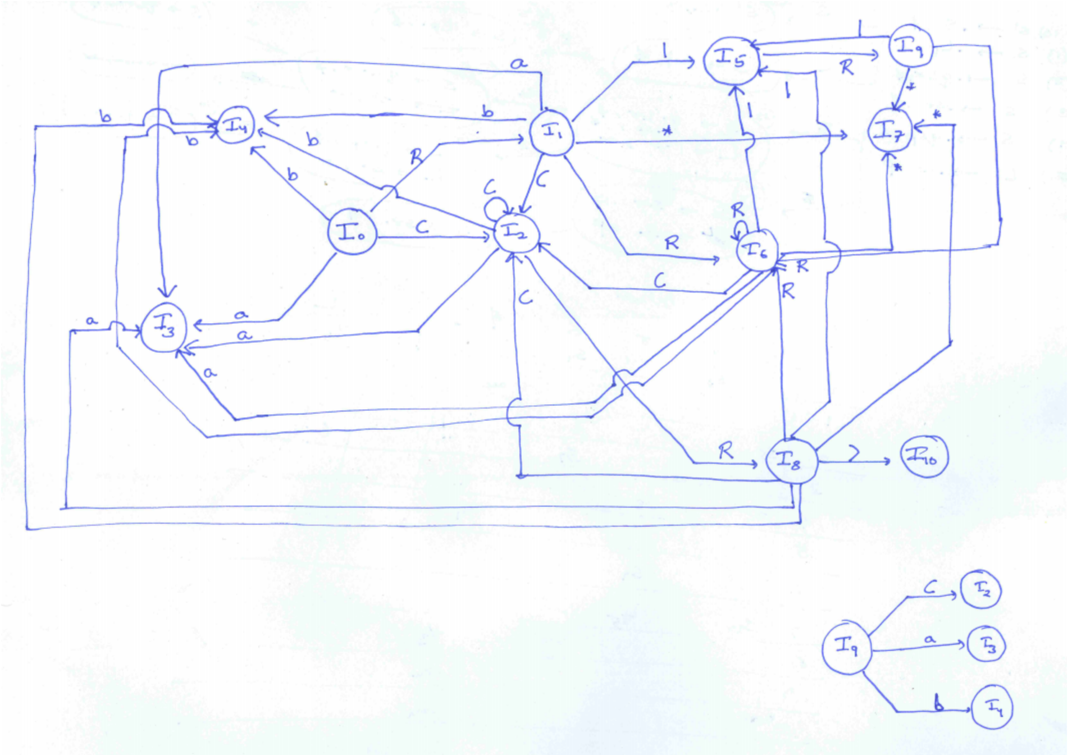
$R \to .RR$
$R \to .R*$
$R \to .(R)$
$R \to .a$
$R \to .b$

$goto(I_5, () = I_2$
$goto(I_5, a) = I_3$
$goto(I_5, b) = I_4$

$goto(I_6, |) = I_5$
$goto(I_6, R) = I_6$
$goto(I_6, *) = I_7$
$goto(I_6, () = I_2$
$goto(I_6, a) = I_3$
$goto(I_6, b) = I_4$

$goto(I_8,)) = I_{10}$
$R \to (R).$
$goto(I_8, |) = I_5$
$goto(I_8, R) = I_6$
$goto(I_8, *) = I_7$
$goto(I_8, () = I_2$
$goto(I_8, a) = I_3$
$goto(I_8, b) = I_4$

$goto(I_9, |) = I_5$
$goto(I_9, R) = I_6$
$goto(I_9, *) = I_7$
$goto(I_9, () = I_2$
$goto(I_9, a) = I_3$
$goto(I_9, b) = I_4$



Now, we proceed to check whether there are any conflicts in the grammar. To do so, we construct the Follow set of R.

$Follow(R) = \{|, *, (,), a, b, \$\}$

We see that for some states like I_6 and I_9, there are shift-reduce conflicts. So, to remove such ambiguities, we use operator precedence and associativity. The operator precedence to eliminate conflicts are as mentioned below:
() has higher precedence than * which has higher precedence than concatenation of RR which has higher precedence than |, i.e.
() > * > concatenation > |

So, using these rules we construct the parsing table as follows,

State	Action	Action	Action	Action	Action	Action	Action	Goto
	|	$*$	(	)	a	b	\$	R
0			s2		s3	s4		1
1	s5	s7	s2		s3	s4	accept	6
2			s2		s3	s4		8
3	r5	r5	r5	r5	r5	r5	r5	
4	r6	r6	r6	r6	r6	r6	r6	
5			s2		s3	s4		9
6	r2	s7	r2	r2	r2	r2	r2	6
7	r3	r3	r3	r3	r3	r3	r3	
8	s5	s7	s2	s10	s3	s4		6
9	r1	s7	s2	r1	s3	s4	r1	6
10	r4	r4	r4	r4	r4	r4	r4	

4. For generating the file lex.yy.c I have used Lex and the code snippets within Lex are written in C++ . Similarly, for the parser which is in test.y, I have written the snippets in C++.
   I have included a script run.sh which

   - Takes an input file
   - Compiles the lex code in words.l into the c file lex.yy.c
   - Compiles the parser in test.y
   - Executes the executable a.out using the grammar and lexer on the input file provided

   To generate the required output file, simply run the command below

   ```
   // The ouput is printed on the standard output
   $ ./run.sh <INPUT-FILE>
   ```