CS335

Assignment II - Flex Bison

Course Instructor :

Dr. Swarnendu Biswas

Submitted By :

Nishant Roshan (200643)

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

February 14, 2023

Problem 1 [50 marks]

For the following grammar, design a predictive parser and show the predictive parsing table. Perform desired processing like removing left-recursion and left-factoring on the grammar if required.

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

$$L \rightarrow L\ ,\ S \mid LS \mid b$$

Solution:

Firstly we transform the given grammar to remove left factoring :

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

$$A \rightarrow ,S \mid S$$

Now again transform the grammar to remove left recursion in rule # 2 :

$$S \rightarrow (L) \mid a \qquad L \rightarrow bL'$$

$$L' \rightarrow AL' \mid \in \qquad A \rightarrow ,S \mid S$$

(Here A and L' are newly introduced non-terminal symbols used in removing left factoring and left recursion respectively)

Calculation of FIRST and FOLLOW sets:

FIRST(S) = { (, a } FOLLOW(S) = { $, , , (, a }

FIRST(L) = { b } FOLLOW(L) = {) }

FIRST(A) = { , , (, a } FOLLOW(A) = { , , (, a }

FIRST(L') = { , , (, a , ∈ } FOLLOW(L') = {) }

PREDECTIVE PARSING TABLE :

Non Terminal	(	)	a	,	b	$
S	S → (L)		S → a			
L					L → bL'	
L'	L' → AL'	L' → ∈	L' → AL'	L' → AL'		
A	A → S		A → S	L' → ,S		

Problem 2 [50 marks]

Show that the following grammar is LALR(1) but not SLR(1)

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

$$L \rightarrow s$$

Solution:

Given rules of the grammar are: (rule numbering)

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

First and Follow set computation:

FIRST(S) = { q , s } FOLLOW(S) = { $ }

FIRST(L) = { s } FOLLOW(L) = { p , r }

Firstly we will prove that the given grammar is not SLR(1)

Canonical collection of LR(0) items:

I_0 = Closure({ $S' \rightarrow \bullet S$ }) ={

$S' \rightarrow \cdot S$ $S \rightarrow \cdot sr$

$S \rightarrow \cdot Lp$ $S \rightarrow \cdot qsp$

$S \rightarrow \cdot qLr$ $L \rightarrow \cdot s$ }

I_1 = Goto (I_0 , S) = {

$S' \rightarrow S \cdot$ }

I_2 = Goto (I_0 , L) = {

$S \rightarrow L \cdot p$ }

I_3 = Goto(I_0 , q) = {

S → q•Lr

L → q•sp

L → •s }

I_4 = Goto(I_0 , s) = {

S → s•r

L → s• }

I_5 = Goto(I_2 , p) = {

S → Lp• }

I_6 = Goto(I_3 , L) = {

S → qL•r }

I_7 = Goto(I_3 , s) = {

S → qs•p

L → s• }

I_8 = Goto(I_4 , r) = {

S → sr• }

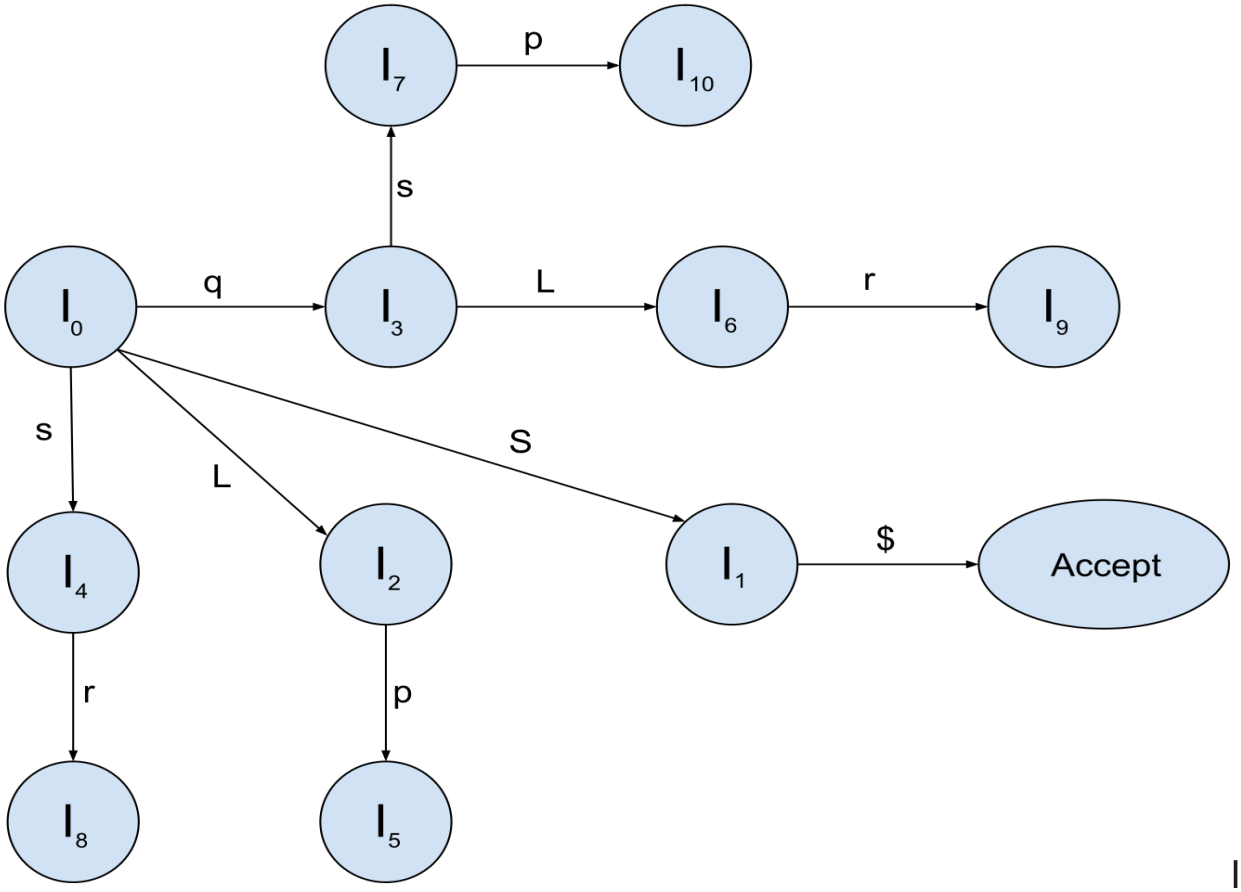
I_9 = Goto(I_6 , r) = {

S → qLr• }

I_{10} = Goto(I_7 , p) = {

S → qsp• }

The LR(0) automaton:



I

The SLR(1) parsing table:

State	Action Table					Goto Table	
	p	q	r	s	$	S	L
0		S3		S4		1	2
1					Accept		
2	S5						
3				S7			6
4	R5		S8 R5				
5					R1		
6			S9				
7	S10 R5		R5				
8					R3		
9					R2		
10					R4		

Here we see that Action[4,r] = Shift 8 OR Reduce 5 (Shift reduce conflict)

Also Action[7,p] = Shift 10 OR Reduce 5 (Shift reduce conflict)

Since SLR(1) parsing table has multiple entries, the SLR parser would not be able to parse the given grammar. Hence the given grammar is not SLR(1).

Now we will prove that the given grammar is LALR(1) :

Canonical collection of LR(1) items:

I_0 = (Closure [S' → •S , $]) = {

S' → •S , $

S' → •Lp , $

S → •qLr , $

S → •sr , $

S → •qsp , $

L → •s , p }

I_1 = Goto(I_0 , S) = {

S' → S• , $ }

I_2 = Goto(I_0 , L) = {

S → L•p , $. }

I_3 = Goto(I_0 , q) = {

S → q•Lr , $.

L → •s , r

S → q•sp , $ }

I_6 = Goto(I_3 , L) = {

S → qL•r , $ }

I_7 = Goto(I_3 , s) = {

L → s• , r

S → qs•p , $ }

I_8 = Goto(I_4 , r) = {

S → sr• , $ }

I_9 = Goto(I_6 , r) = {

I_4 = Goto(I_0 , s) = {

S → s•r , $

L → s• , p }

I_5 = Goto(I_2 , p) = {

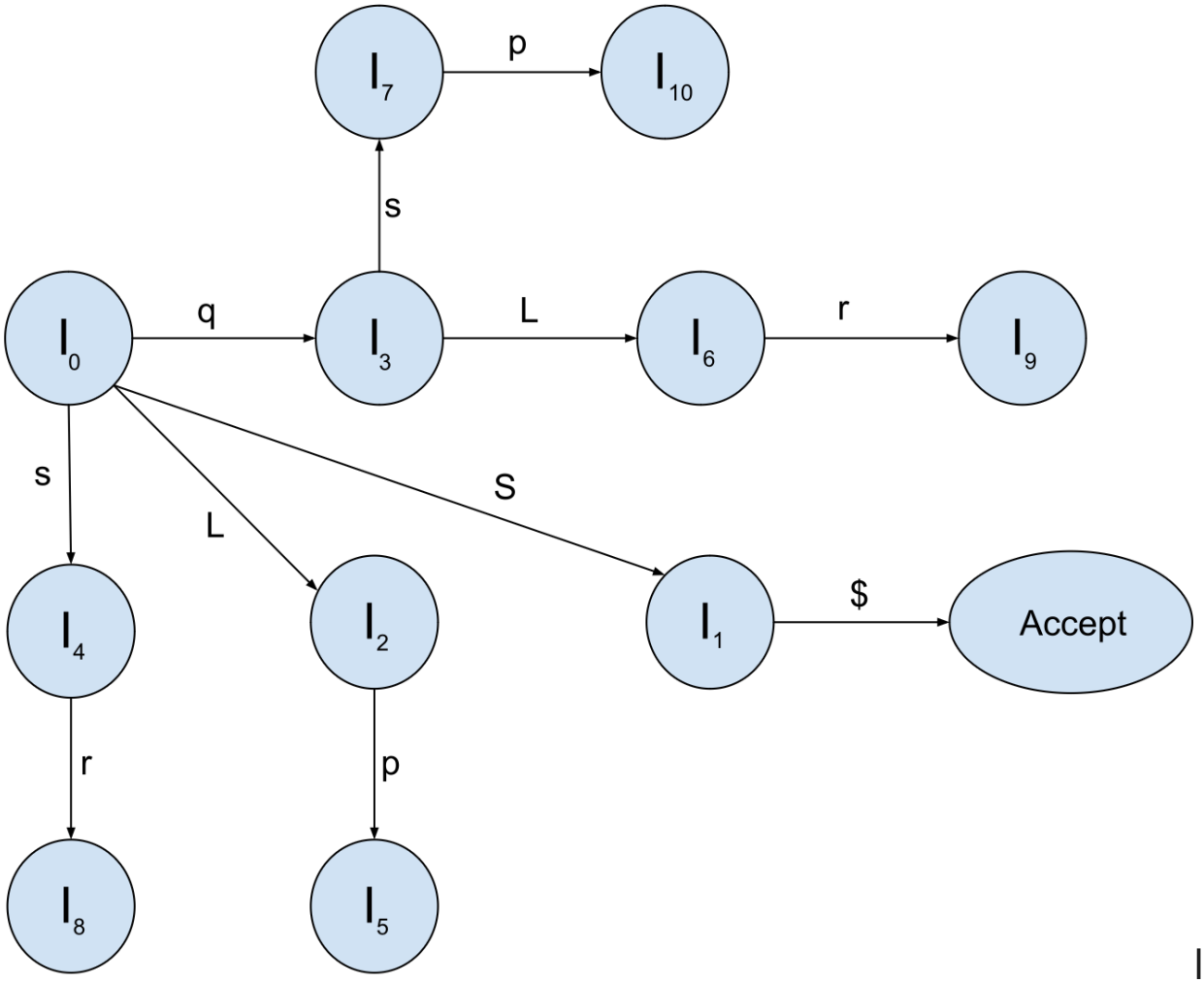
S → Lp• , $ }

S → qLr• , $ }

I_{10} = Goto(I_7 , p) = {

S → qsp• , $ }

The LR(1) automaton:



The LALR parsing table:

State	Action Table					Goto Table	
	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		

Problem 3 [50 marks]

Construct an SLR parsing table for the following grammar. Show the canonical set of states and the transition diagram.

R → R '|' R
R → RR
R → R*
R → (R)
R → a | b

Note that the vertical bar in the first production is the "or" symbol (i.e., terminal), and is not a separator between alternations.

Resolve the parsing action conflicts in such a way that regular expressions will be parsed normally. Include your disambiguation rules in the PDF file, and show the final parsing table.

Solution:

Introduce a new start state S, such that we add a rule S → R to the grammar. The numbering of rules would then be:

1) S → R
2) R → R '|' R
3) R → RR
4) R → R*
5) R → (R)
6) R → a
7) R → b

First and follow set calculation:

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

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

Canonical Collection of LR(0) items:

I_0 = Closure ({S → •R}) = {

S → •R
R → •R '|' R.
R → •RR
R → •R*
R → •(R)
R → •a
R → •b
}

I_1 = Goto(I_0 , R) = {

S → R•
R → R•'|'R
R → R•R
R → R•*
R → •R'|'R
R → •RR
R → •R*
R → •(R)
R → •a
R→ •b
}

I_2 = Goto(I_0 , '(') = {

R → (•R)
R → •R '|' R
R → •RR
R → •R*
R → •(R)
R → •a
R → •b
}

I_3 = Goto(I_0 , a) = {

R → a•
}

I_4 = Goto(I_0 , b) = {

R → b•
}

$I_5 = Goto(I_1 , R) = \{$

$R \rightarrow RR\bullet$ $R \rightarrow \bullet RR$

$R \rightarrow R\bullet \text{ '|' } R$ $R \rightarrow \bullet R^*$

$R \rightarrow R\bullet R$ $R \rightarrow \bullet(R)$

$R \rightarrow R\bullet *$ $R \rightarrow \bullet a$

$R \rightarrow \bullet R \text{ '|' } R$ $R \rightarrow \bullet b \quad \}$

$I_6 = Goto(I_1 , \text{'|'}) = \{$

$R \rightarrow R \text{ '|' } \bullet R.$ $R \rightarrow \bullet(R)$

$R \rightarrow \bullet R \text{ '|' } R$ $R \rightarrow \bullet a$

$R \rightarrow \bullet RR$ $R \rightarrow \bullet b$

$R \rightarrow \bullet R^*$ $\}$

$I_7 = Goto(I_1 , *) = \{$

$R \rightarrow R^*\bullet$ $\}$

$I_8 = Goto(I_2 , R) = \{$

$R \rightarrow (R\bullet)$ $R \rightarrow \bullet RR$

$R \rightarrow R\bullet \text{ '|' } R$ $R \rightarrow \bullet R^*$

$R \rightarrow R\bullet R$ $R \rightarrow \bullet(R)$

$R \rightarrow R\bullet *$ $R \rightarrow \bullet a$

$R \rightarrow \bullet R \text{ '|' } R$ $R \rightarrow \bullet b \quad \}$

$I_9 = Goto(I_6 , R) = \{$

$R \rightarrow R \text{ '|' } R\bullet$ $R \rightarrow \bullet RR$

$R \rightarrow R \bullet \text{ '|' } R$ $R \rightarrow \bullet R^*$

$R \rightarrow R\bullet R$ $R \rightarrow \bullet(R)$

$R \rightarrow R\bullet *$ $R \rightarrow \bullet a$

$R \rightarrow \bullet R \text{ '|' } R$ $R \rightarrow \bullet b \quad \}$

$I_2 = Goto(I_1 , \text{'('})$

$I_3 = Goto(I_1 , a)$

$I_4 = Goto(I_1 , b)$

$I_2 = Goto(I_2 , \text{'('})$

$I_3 = Goto(I_2 , a)$

$I_4 = Goto(I_2 , b)$

$I_5 = Goto(I_5 , R)$

$I_6 = Goto(I_5 , |)$

$I_7 = Goto(I_5 , *)$

$I_8 = Goto(I_5 , \text{'('})$

$I_3 = Goto(I_5 , a)$

$I_4 = Goto(I_5 , b)$

I_2 = Goto(I_6 , '(')

I_3 = Goto(I_6 , a)

I_4 = Goto(I_6 , b)

I_5 = Goto(I_8 , R)

I_6 = Goto(I_8 , '|')

I_7 = Goto(I_8 , *)

I_2 = Goto(I_8 , '(')

I_{10} = Goto(I_8 , ')') ={

R → (R)• }

I_3 = Goto(I_8 , a)

I_4 = Goto(I_8 , b)

I_5 = Goto(I_9 , R)

I_6 = Goto(I_9 , '|')

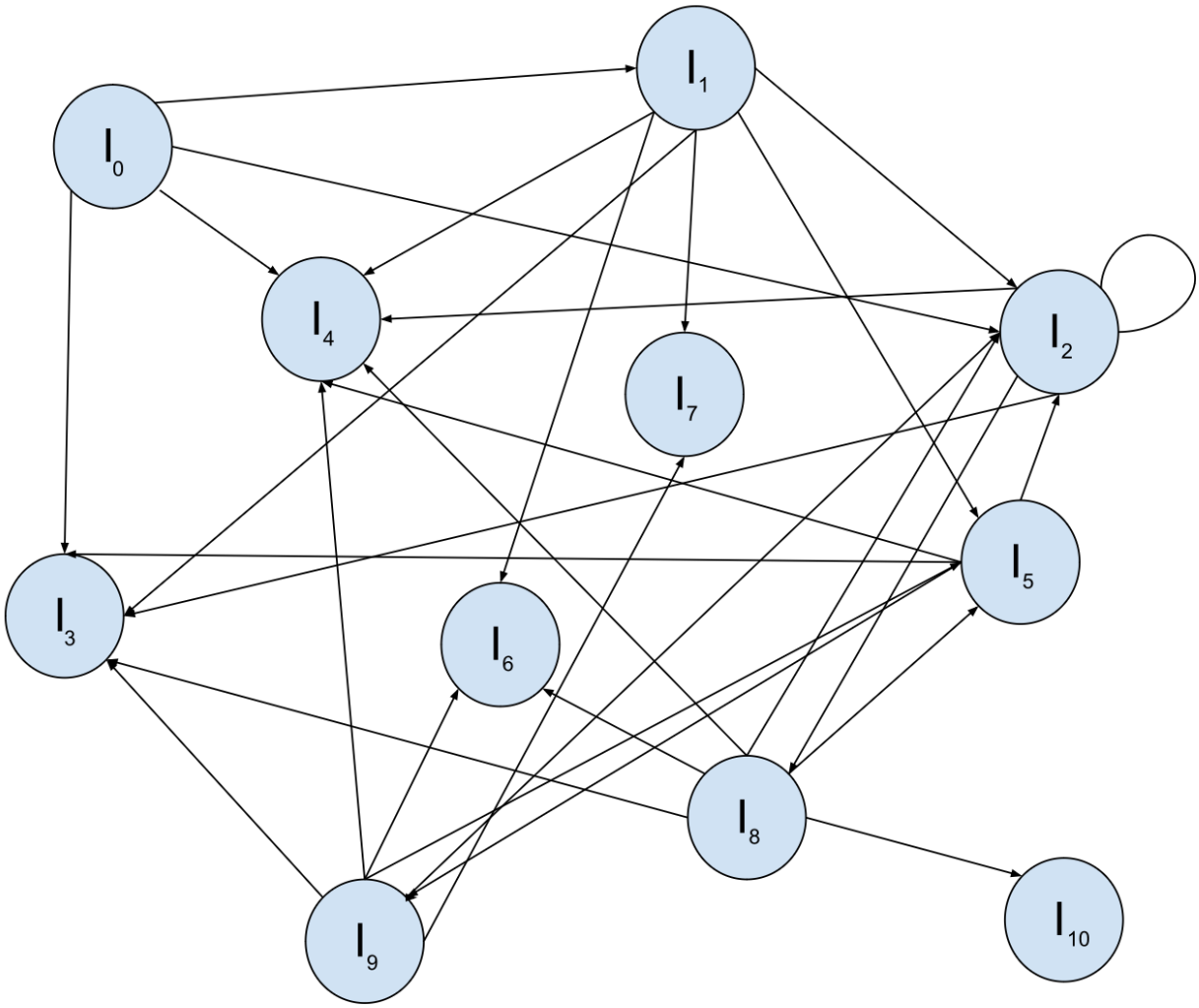
I_2 = Goto(I_9 , '(')

I_7 = Goto(I_9 , *)

I_3 = Goto(I_9 , a)

I_4 = Goto(I_9 , b)

The LR(0) automaton for this grammar would be:



The SLR parsing table:

States	*	'|'	)	(	a	b	$	R
0				S2	S3	S4		1
1	S7	S6		S2	S3	S4	accept	5
2				S2	S3	S4		8
3	R6	R6	R6	R6	R6	R6	R6	
4	R7	R7	R7	R7	R7	R7	R7	
5	R3,S7	R3,S6	R3	R3,S2	R3,S3	R3,S4	R3	
6				S2	S3	S4		9

7	R4	R4	R4	R4	R4	R4	R4	
8	S7	S6	S10	S2	S3	S4		5
9	R2,S7	R2,S6	R2	R2,S2	R2,S3	R2,S4	R2	5
10	R5	R5	R5	R5	R5	R5	R5	

Now, to resolve the shift reduce conflicts in the above parsing table,

Let us assume the precedence order among the productions:

$$(R) > R^* > RR > R'|'R$$

I have assumed this precedence order, since it is the same as that of regex.

Problem 4 [50 marks]

Solution:

Tools used:

A simple combination of Flex and bison. Flex has been used for producing tokens and passing it to the bison program for further parsing. Bison parses the grammar based on tokens returned.

Name of Flex file: parser.l

Name of Bison file: parser.y

To run the file:

1) Make sure that both flex and bison are installed in the system.
2) Put the Makefile, lex file (parser.l) , bison file (parser.y) and the sample thesis in the same folder.

3) Now open terminal in that folder and simple run the command 'make' to compile all the files.

   Alternatively, run the following commands to compile:

   bison -t -v -d parser.y

   flex parser.l

   g++ -o parser parser.tab.c lex.yy.c

4) Run the command: ./parser name_of_thesis_file to produce the final output.