ASSIGNMENT 2

KESHAV BANSAL(170335)

CS335

Question

1. **For the following grammar, design a predictive parser and show the predictive parsing table. Remove left-recursion (if any), left-factor your grammar.**
   $S \rightarrow (L) \mid a$
   $L \rightarrow L, S \mid LS \mid b$

   Solution

   Grammar after removing left recursion:
   $S \rightarrow (L) \mid a$
   $L \rightarrow bX$
   $X \rightarrow , SX \mid SX \mid \epsilon$
   First and follow set computation :

   First$(L) = \{\ b\ \}$

   First$(S) = \{\ (\ , a\ \}$

   First$(X) = \{\ $‘,’ , $(\ , a\ , \epsilon\ \}$

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

   Follow$(S) = \{\ $‘,’ , $(\ , a\ ,$ \$, $)\ \}$

   Follow$(X) = \{\ )\ \}$

   Note: ‘,’ is used to denote the terminal comma(,) in the grammar

   Predictive parsing table :

Nonterminal	(	)	‘,’	a	b	$
S	$S \rightarrow (L)$			$S \rightarrow a$		
L					$L \rightarrow bX$	
X	$X \rightarrow SX$	$X \rightarrow \epsilon$	$X \rightarrow , SX$	$X \rightarrow SX$		

Question

2. **Show that the given grammar is LALR(1) but not SLR(1).**

Adding a dummy start state S' we get the following rules :

```
1) S' → S
2) S → Lp
3) S → qLr
4) S → sr
5) S → qsp
6) L → s
```

Canonical states :

I_0 =

```
S' → .S
S → .Lp
S → .qLr
S → .sr
S → .qsp
L → .s
```

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
S → 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.
```

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

STATE	**ACTION**					**GOTO**		
	p	q	r	s	$	S'	S	L
0		s3		s4			1	2
1					accept			
2	s5							
3				s7				6
4	r6		r6 s8					
5					r2			
6			s9					
7	r6 s10		r6					
8					r4			
9					r3			
10					r5			

Table 1: Parse table for SLR(1)

Since there are 2 shift reduce conficts, the grammar is SLR(1) ambiguous.

Canonical states for LALR(1):

I_0 =

| 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 , $
| S → q.sp , $
| L → .s , *r*

I_4 = goto(I_0,s) =

| S → s.r , $
| L → s. , *p*

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. , *r*

I_8 = goto(I_4,r) =

| S → sr. , $

I_9 = goto(I_6,r) =

| S → qLr. , $

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

| S → qsp. , $

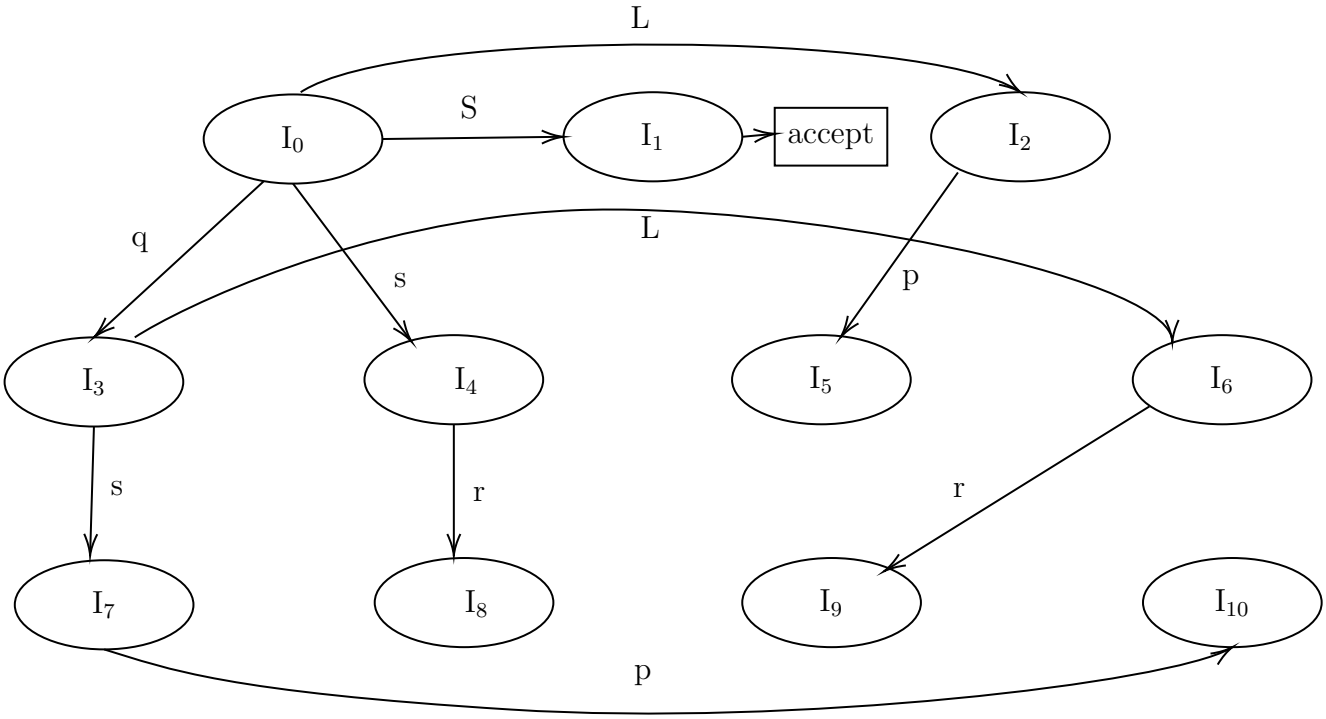

Figure 1: State diagram for SLR and LALR parser

STATE	**ACTION**					**GOTO**		
	p	q	r	s	$	S'	S	L
0		s3		s4			1	2
1					accept			
2	s5							
3				s7				6
4	r6		s8					
5					r2			
6			s9					
7	s10		r6					
8					r4			
9					r3			
10					r5			

Table 2: Parse table for LALR(1)

Since each cell contains atmost one action, the grammar is LALR(1)

Question

3. **Construct an SLR parsing table for the given grammar. Resolve the parsing action conflicts in such a way that regular expressions will be parsed normally.**

Adding a dummy start state S we get the following rules :

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

Canonical states

I_0 =

```
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|.R
R → .R|R
R → .RR
R → .R*
R → .(R)
R → .a
R → .b
```

I_6 = goto(I_1,$*$) =

```
R → R*.
```

I_7 = goto(I_1,R) =

```
R → RR.
R → R.|R
R → R.R
R → R.*
R → .R|R
R → .RR
R → .R*
R → .(R)
R → .a
R → .b
```

I_8 = goto(I_2,R) =

```
R → (R.)
R → R.|R
R → R.R
R → R.*
R → .R|R
R → .RR
R → .(R)
R → .a
R → .b
```

I_9 = goto(I_5,R) =

```
R → R|R.
R → R.|R
R → R.R
R → R.*
R → .R|R
R → .RR
R → .R*
R → .(R)
R → .a
R → .b
```

I_{10} = goto(I_8,)) = | | R → (R) .

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

STATE	**ACTION**							**GOTO**	
	|	*	(	)	a	b	$	S	R
0			s2		s3	s4			1
1	s5	s6	s2		s3	s4	accept		7
2			s2		s3	s4			8
3	r6	r6	r6	r6	r6	r6	r6		
4	r7	r7	r7	r7	r7	r7	r7		
5			s2		s3	s4			9
6	r4	r4	r4	r4	r4	r4	r4		
7	r3 s5	r3 s6	r3 s2	r3	r3 s3	r3 s4	r3		7
8	s5	s6	s2	s10	s3	s4			7
9	r2 s5	r2 s6	r2 s2	r2	r2 s3	r2 s4	r2		7
10	r5	r5	r5	r5	r5	r5	r5		

Table 3: Parse table for SLR(1) before disambiguation

As we can see, there are 4 shift reduce conflicts. These arise due to undefined precedence and associativity of | , concatenation and ∗. To solve the conflicts, we follow the following disambiguation rules -

1. **Precedence of * > precedence of concatenation > precedence of |**
2. **| and concatenation are left associative**

The following steps are taken to disambiguate -

- **Row 7 Terminal | :** Precedence of concatenation > Precedence of | hence r3
- **Row 7 Terminal ∗ :** Precedence of ∗ > Precedence of concatenation hence s6
- **Row 7 Terminal (:** concatenation is left associative hence r3
- **Row 7 Terminal a :** concatenation is left associative hence r3
- **Row 7 Terminal b :** concatenation is left associative hence r3
- **Row 9 Terminal | :** | is left associative hence r2
- **Row 9 Terminal ∗ :** Precedence of ∗ > Precedence of | hence s6
- **Row 9 Terminal (:** Precedence of concatenation > Precedence of | hence s2
- **Row 9 Terminal a :** Precedence of concatenation > Precedence of | hence s3
- **Row 9 Terminal b :** Precedence of concatenation > Precedence of | hence s4

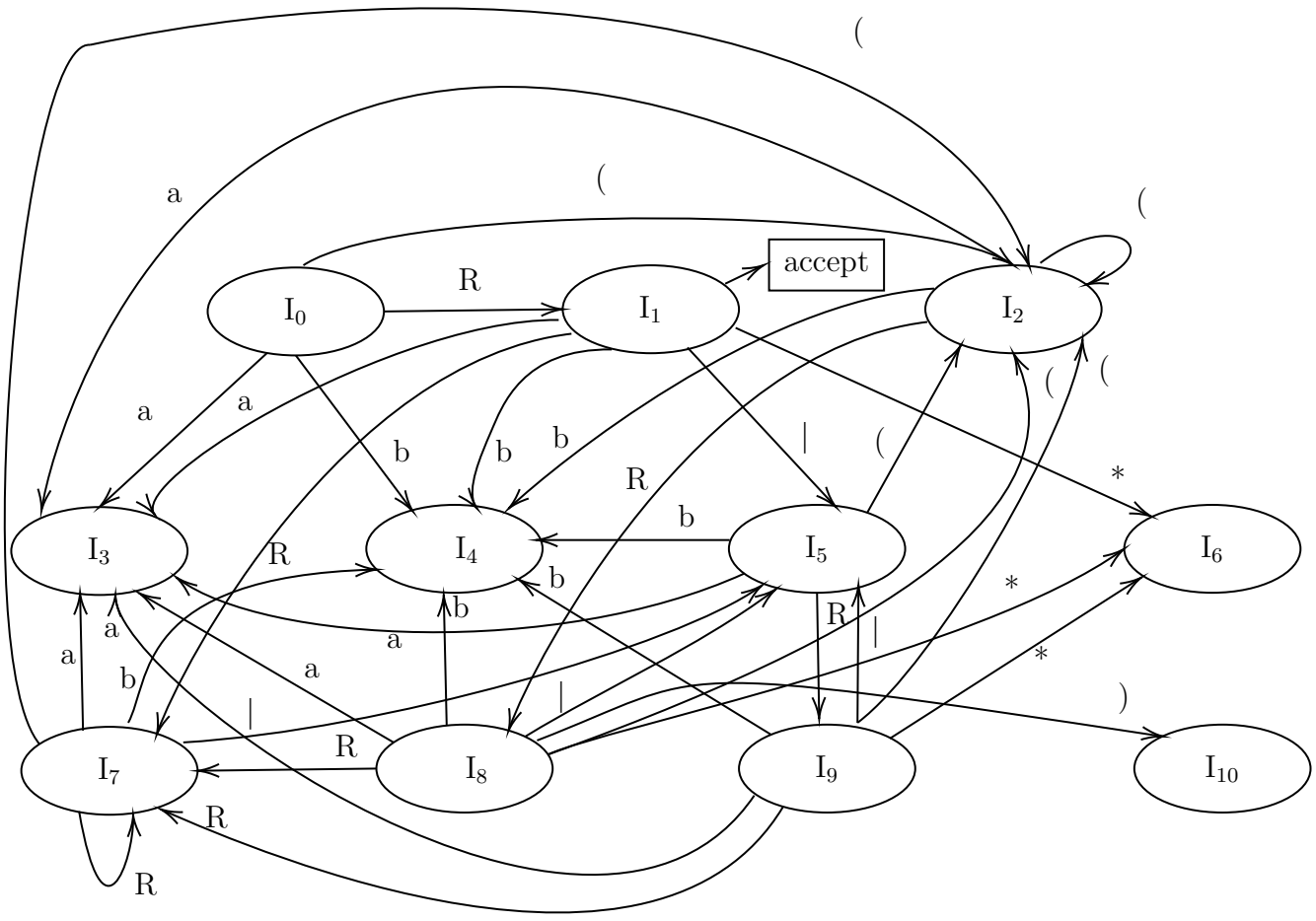

Figure 2: SLR Parsing table

After disambiguation

STATE	ACTION							GOTO	
	|	*	(	)	a	b	$	S	R
0					s3	s4			1
1	s5	s6	s2		s3	s4	accept		7
2			s2		s3	s4			8
3	r6	r6	r6	r6	r6	r6	r6		
4	r7	r7	r7	r7	r7	r7	r7		
5			s2		s3	s4			9
6	r4	r4	r4	r4	r4	r4	r4		
7	r3	s6	r3	r3	r3	r3	r3		7
8	s5	s6	s2	s10	s3	s4			7
9	r2	s6	s2	r2	s3	s4	r2		7
10	r5	r5	r5	r5	r5	r5	r5		

Table 4: Parse table for SLR(1) after disambiguation

Question

4. **Dissertation Parser**

Tools used :

- flex 2.6.4
- bison (GNU Bison) 3.0.4
- g++ 7.4.0

To run the parser, run the following program :

```
bash run.sh <INPUT_FILE>
```

Example:

```
bash run.sh samplethesis.txt
```

The output shows in **STDOU**
The errors will be redirected to **STDERR**.