CS335 Assignment - 2

Aditya Tanwar
200057

February 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.

$$S \to (L) \mid a$$
$$L \to L, S \mid LS \mid b$$

Solution

Firstly, to remove the (direct) left recursion, a new terminal L' is introduced. The process followed is similar to the one described in the lectures. The augmented set of production rules are as follows:

$$S \to (L) \tag{1a}$$
$$S \to a \tag{1b}$$
$$L \to bL' \tag{1c}$$
$$L' \to, SL' \tag{1d}$$
$$L' \to SL' \tag{1e}$$
$$L' \to \epsilon \tag{1f}$$

There is no Left Recursion in the above production rules, neither direct nor indirect, thus we move on to the next step, construction of FIRST and FOLLOW sets. The terminals have been written in gray to avoid ambiguity due to characters like commas, parentheses, etc.

- FIRST sets: The FIRST sets of terminals have been omitted since they'll only be containing the terminal itself.

$$\text{FIRST}(S) = \{(, a\}$$
$$\text{FIRST}(L) = \{b\}$$
$$\text{FIRST}(L') = \{,, (, a, \epsilon\}$$

- FOLLOW sets:

$$\begin{aligned}
&\text{FOLLOW}(S) \supseteq \text{FIRST}(L')\backslash\{\epsilon\} && \text{(From (1d) and (1f))}\\
&\text{FOLLOW}(S) = \{\$, ,, (,), a\}\\
&\text{FOLLOW}(L) = \{)\} && \text{(From (1a))}\\
&\text{FOLLOW}(L') \supseteq \text{FOLLOW}(L) && \text{(From (1c))}\\
&\text{FOLLOW}(L') = \{)\}
\end{aligned}$$

Equipped with these sets, the table for the predictive parser can be constructed. Empty entries in the table imply an error, in line with the tables in lecture slides. The table is:

State	(	a	,	b	)	\$
S	$S \to (L)$	$S \to a$				
L				$L \to bL'$		
L'	$L' \to SL'$	$L' \to SL'$	$L' \to ,SL'$			$L' \to \epsilon$

Problem 2

[50 marks]

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

$$\begin{aligned}
&S \to Lp \mid qLr \mid sr \mid qsp\\
&L \to s
\end{aligned}$$

2.1 Solution

It is first shown that the grammar is not SLR(1) by constructing the parsing table and showing that it contains a conflict. Later, an LALR(1) parser is made for the language to show that the grammar is indeed LALR(1). Firstly, a "dummy" Non-terminal S' is added and the production rules are re-written as follows:

$$\begin{aligned}
&S' \to S && \text{(2a)}\\
&S \to Lp && \text{(2b)}\\
&S \to qLr && \text{(2c)}\\
&S \to sr && \text{(2d)}\\
&S \to qsp && \text{(2e)}\\
&L \to s && \text{(2f)}
\end{aligned}$$

2.1.1 SLR(1)

To start with the parser, firstly, the FIRST and FOLLOW sets are formed, followed by the canonical collection of sets of LR(0) items:

- FIRST sets:

$$\begin{aligned}
&\text{FIRST}(L) = \{s\} && \text{(From (2f))}\\
&\text{FIRST}(S) \supseteq \text{FIRST}(L) && \text{(From (2b))}\\
&\text{FIRST}(S) = \{s, q\}\\
&\text{FIRST}(S') = \text{FIRST}(S) = \{s, q\} && \text{(From (2a))}
\end{aligned}$$

- FOLLOW sets:

$$\begin{aligned}
&\text{FOLLOW}(S) = \{\$\}\\
&\text{FOLLOW}(L) = \{p, r\} && \text{(From (2b) and (2c) respectively)}
\end{aligned}$$

- Canonical collection of sets using CLOSURE and GOTO:

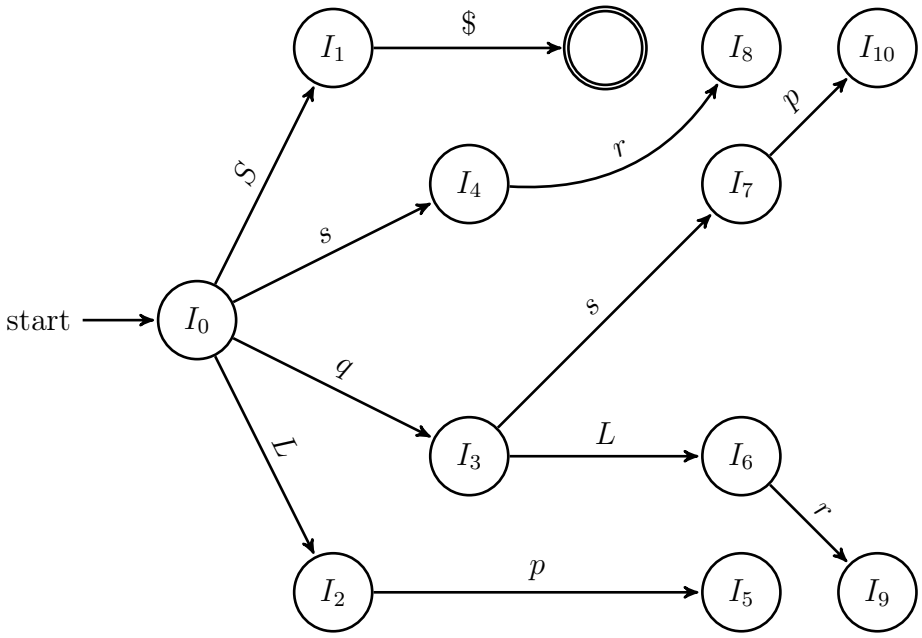
$$\begin{aligned}
I_0 &= \text{CLOSURE}(\{S' \to \bullet S\})\\
&= \{S' \to \bullet S\} \cup \{S \to \bullet Lp\} \cup \{S \to \bullet qLr\} \cup \{S \to \bullet sr\} \cup \{S \to \bullet qsp\} \cup \{L \to \bullet s\}\\
I_1 &= \text{GOTO}(I_0, S) = \{S' \to S\bullet\}\\
I_2 &= \text{GOTO}(I_0, L) = \{S \to L \bullet p\}\\
I_3 &= \text{GOTO}(I_0, q) = \{S \to q \bullet Lr\} \cup \{S \to q \bullet sp\} \cup \{L \to \bullet s\}\\
I_4 &= \text{GOTO}(I_0, s) = \{S \to s \bullet r\} \cup \{L \to s\bullet\}\\
I_5 &= \text{GOTO}(I_2, p) = \{S \to Lp\bullet\}\\
I_6 &= \text{GOTO}(I_3, L) = \{S \to qL \bullet r\}\\
I_7 &= \text{GOTO}(I_3, s) = \{S \to qs \bullet p\} \cup \{L \to s\bullet\}\\
I_8 &= \text{GOTO}(I_4, r) = \{S \to sr\bullet\}\\
I_9 &= \text{GOTO}(I_6, r) = \{S \to qLr\bullet\}\\
I_{10} &= \text{GOTO}(I_7, p) = \{S \to qsp\bullet\}
\end{aligned}$$

Now, equipped with these entities, the parser table is constructed as follows:

- s_i denotes transition to state i, and r_j denotes reduction using j^{th} production rule.
- Empty cells imply an error.
- Filling the table with shift actions is straightforward, especially since there are transitions only from a state i to state j where $i < j$.
- As for the reduction rules, whenever a set has an item of the kind $(A \to \alpha\bullet)$, a reduction rule is added to the corresponding state in cells corresponding to each a in FOLLOW(A).

State	Action					GOTO	
	p	q	r	s	\$	S	L
0		s_3		s_4		1	2
1					*accept*		
2	s_5						
3				s_7			6
4	$r_{(2f)}$		$s_8/r_{(2f)}$				
5					$r_{(2b)}$		
6			s_9				
7	$s_{10}/r_{(2f)}$		$r_{(2f)}$				
8					$r_{(2d)}$		
9					$r_{(2c)}$		
10					$r_{(2e)}$		

As can be seen, there are two *shift-reduce conflicts*, arising in the states (4) and (7). Thus, it follows that the grammar is not SLR(1), since such a parser would have conflicts. Regardless, an automaton corresponding to the parser table has been drawn below:



2.1.2 LALR(1)

To construct an LALR(1) parser, canonical collection of sets of LR(1) items is formed, since FIRST and FOLLOW sets remain the same as formed in SLR(1). The same definition, as the one provided in the lecture slides is used, with FIRST(\$)={\$}

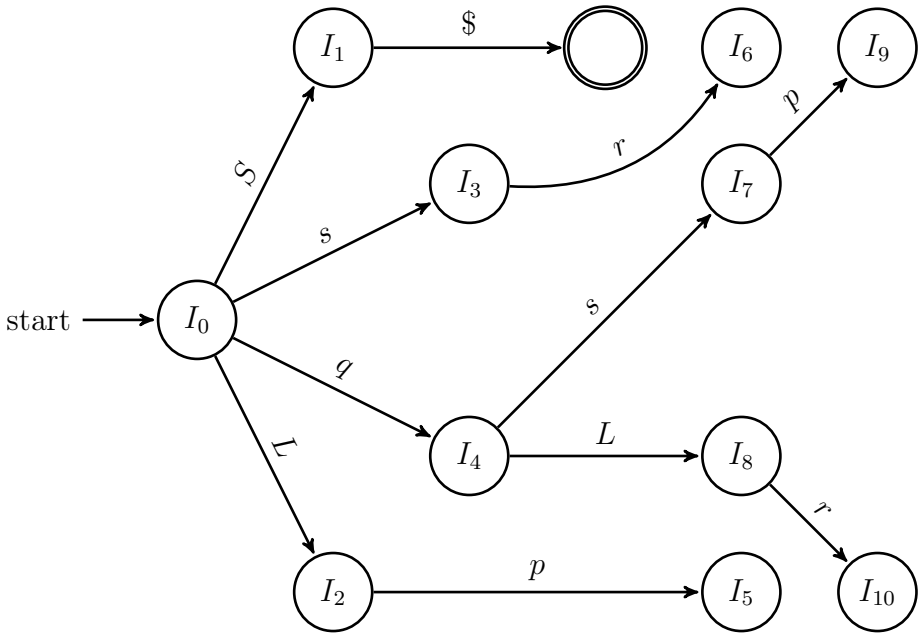
$$
\begin{aligned}
I_0 &= \text{CLOSURE}([S' \to \bullet S, \$]) \\
&= \{[S' \to \bullet S, \$],\ [S \to \bullet Lp, \$],\ [L \to \bullet s, p],\ [S \to \bullet qsp, \$],\ [S \to \bullet qLr, \$],\ [S \to \bullet sr, \$]\} \\
I_1 &= \text{GOTO}(I_0, S) = \{[S' \to S\bullet, \$]\} \\
I_2 &= \text{GOTO}(I_0, L) = \{[S \to L \bullet p, \$]\} \\
I_3 &= \text{GOTO}(I_0, s) = \{[L \to s\bullet, p],\ [S \to s \bullet r, \$]\} \\
I_4 &= \text{GOTO}(I_0, q) = \{[S \to q \bullet sp, \$],\ [S \to q \bullet Lr, \$],\ [L \to \bullet s, r]\} \\
I_5 &= \text{GOTO}(I_2, p) = \{[S \to Lp\bullet, \$]\} \\
I_6 &= \text{GOTO}(I_3, r) = \{[S \to sr\bullet, \$]\} \\
I_7 &= \text{GOTO}(I_4, s) = \{[S \to qs \bullet p, \$],\ [L \to s\bullet, r]\} \\
I_8 &= \text{GOTO}(I_4, L) = \{[S \to qL \bullet r, \$]\} \\
I_9 &= \text{GOTO}(I_7, p) = \{[S \to qsp\bullet, \$]\} \\
I_{10} &= \text{GOTO}(I_8, r) = \{[S \to qLr\bullet, \$]\}
\end{aligned}
$$

Since no two I_i subsets have the same "core" (i.e., LR(0) items), there is no need to merge any two sets in order to use these LR(1) items for an LALR(1) parser. We continue with these sets as is, instead of introducing J_i sets. In essence, the parser's table can be directly constructed with this canonical collection of sets of LR(1) items. The table is constructed as follows:

- s_i denotes transition to state i, and r_j denotes reduction using j^{th} production rule.
- Empty cells imply an error.
- Filling the table with shift actions is straightforward, especially since there are transitions only from a state i to state j where $i < j$.
- As for the reduction rules, whenever a set has an item of the kind $[A \to \alpha\bullet, a]$, a reduction rule is added to the corresponding state in the cell corresponding to a.

State	Action					GOTO	
	p	q	r	s	\$	S	L
0		s_4		s_3		1	2
1					*accept*		
2	s_5						
3	$r_{(2f)}$		s_6				
4				s_7			8
5					$r_{(2b)}$		
6					$r_{(2d)}$		
7	s_9		$r_{(2f)}$				
8			s_{10}				
9					$r_{(2e)}$		
10					$r_{(2c)}$		

Since, there are no conflicts in the LALR(1) parser's table, thus we conclude that the language is LALR(1). However, from the previous section, we know that the language is not SLR(1).



Problem 3

[50 marks]

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

$$
\begin{aligned}
R &\to R\ '|'\ R \\
R &\to RR \\
R &\to R^* \\
R &\to (R) \\
R &\to a \mid b
\end{aligned}
$$

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

We construct an SLR(1) parser for this language. We first number each production rule after adding $R' \to R$, followed by finding the FIRST and FOLLOW sets of R (terminals are written in gray to avoid ambiguity):

$$
\begin{aligned}
R' &\to R && \text{(3a)} \\
R &\to R|R && \text{(3b)} \\
R &\to RR && \text{(3c)} \\
R &\to R^* && \text{(3d)} \\
R &\to (R) && \text{(3e)} \\
R &\to a && \text{(3f)} \\
R &\to b && \text{(3g)} \\
\text{FIRST}(R) &= \{(, a, b, \} \\
\text{FOLLOW}(R) &= \{(,), a, b, *, |, \$\}
\end{aligned}
$$

With this done, we move on to constructing canonical collection sets of LR(0) items. In addition to the standard I_i sets in the lecture slides, we also make two auxiliary sets R_0 and R_1 for compact notation. Again, terminals have been written in gray to avoid ambiguity.

$$
\begin{aligned}
R_0 &\coloneqq \text{CLOSURE}(\{R \to \bullet RR\}) \\
&= \{R \to \bullet R|R\} \cup \{R \to \bullet RR\} \cup \{R \to \bullet R^*\} \cup \{R \to \bullet(R)\} \cup \{R \to \bullet a\} \cup \{R \to \bullet b\} \\
R_1 &\coloneqq \text{GOTO}(R_0, R) \\
&= \{R \to R \bullet |R\} \cup \{R \to R \bullet R\} \cup \{R \to R \bullet {}^*\} \cup R_0 \\
I_0 &= \text{CLOSURE}(\{R' \to \bullet R\}) = \{R' \to \bullet R\} \cup R_0 \\
I_1 &= \text{GOTO}(I_0, R) = \{R' \to R\bullet\} \cup R_1 \\
I_2 &= \text{GOTO}(I_0, () = \{R \to (\bullet R)\} \cup R_0
\end{aligned}
$$

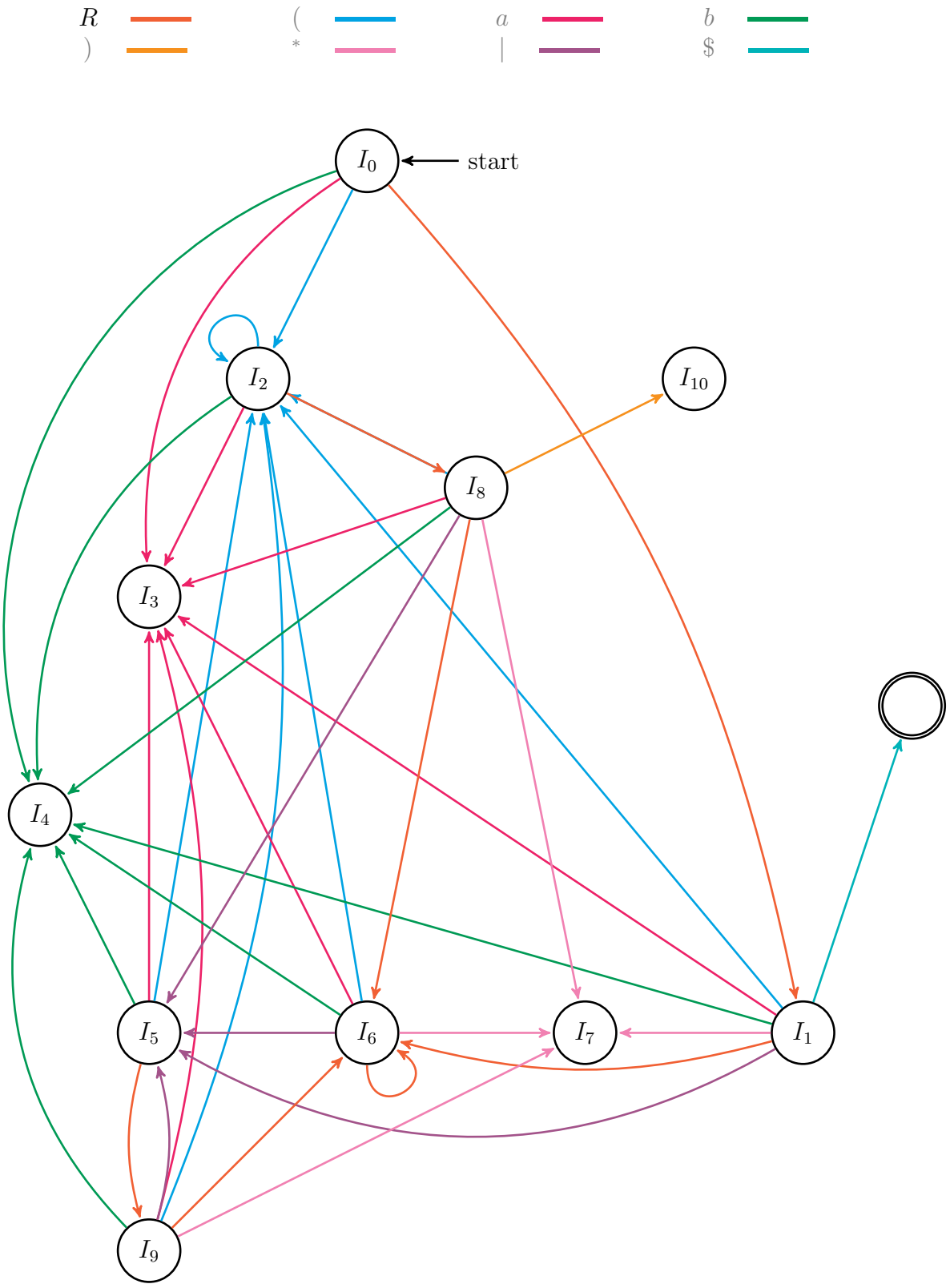
$$
\begin{aligned}
I_3 &= \text{GOTO}(I_0, a) = \{R \to a\bullet\} \\
I_4 &= \text{GOTO}(I_0, b) = \{R \to b\bullet\} \\
I_5 &= \text{GOTO}(I_1, |) = \{R \to R| \bullet R\} \cup R_0 \\
I_6 &= \text{GOTO}(I_1, R) = \{R \to RR\bullet\} \cup R_1 \\
I_7 &= \text{GOTO}(I_1, {}^*) = \{R \to R^*\bullet\} \\
I_8 &= \text{GOTO}(I_2, R) = \{R \to (R \bullet)\} \cup R_1 \\
I_9 &= \text{GOTO}(I_5, R) = \{R \to R|R\bullet\} \cup R_1 \\
I_{10} &= \text{GOTO}(I_8,)) = \{R \to (R)\bullet\}
\end{aligned}
$$

$$
\begin{array}{lll}
I_2 = \text{GOTO}(I_1, () & I_3 = \text{GOTO}(I_1, a) & I_4 = \text{GOTO}(I_1, b) \\
I_2 = \text{GOTO}(I_2, () & I_3 = \text{GOTO}(I_2, a) & I_4 = \text{GOTO}(I_2, b) \\
I_2 = \text{GOTO}(I_5, () & I_3 = \text{GOTO}(I_5, a) & I_4 = \text{GOTO}(I_5, b) \\
I_2 = \text{GOTO}(I_6, () & I_3 = \text{GOTO}(I_6, a) & I_4 = \text{GOTO}(I_6, b) \\
I_5 = \text{GOTO}(I_6, |) & I_6 = \text{GOTO}(I_6, R) & I_7 = \text{GOTO}(I_6, {}^*) \\
I_2 = \text{GOTO}(I_8, () & I_3 = \text{GOTO}(I_8, a) & I_4 = \text{GOTO}(I_8, b) \\
I_5 = \text{GOTO}(I_8, |) & I_6 = \text{GOTO}(I_8, R) & I_7 = \text{GOTO}(I_8, {}^*) \\
I_2 = \text{GOTO}(I_9, () & I_3 = \text{GOTO}(I_9, a) & I_4 = \text{GOTO}(I_9, b) \\
I_5 = \text{GOTO}(I_9, |) & I_6 = \text{GOTO}(I_9, R) & I_7 = \text{GOTO}(I_9, {}^*)
\end{array}
$$

The SLR(1) parsing table made with these LR(0) items is given below (same rules have been followed as before.

State	Action							GOTO
	(	|	)	*	a	b	$	R
0	s_2				s_3	s_4		1
1	s_2	s_5		s_7	s_3	s_4	*accept*	6
2	s_2				s_3	s_4		8
3	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	
4	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	
5	s_2				s_3	s_4		9
6	$s_2/r_{(3c)}$	$s_5/r_{(3c)}$	$r_{(3c)}$	$s_7/r_{(3c)}$	$s_3/r_{(3c)}$	$s_4/r_{(3c)}$	$r_{(3c)}$	6
7	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	
8	s_2	s_5	s_{10}	s_7	s_3	s_4		6
9	$s_2/r_{(3b)}$	$s_5/r_{(3b)}$	$r_{(3b)}$	$s_7/r_{(3b)}$	$s_3/r_{(3b)}$	$s_4/r_{(3b)}$	$r_{(3b)}$	6
10	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	

Using the above table, we make an automaton. Due to the number of states, and density of the "graph", the transition terms cannot be written on the arrows. Thus, they have instead been colour coded with the following legend:



Before discussing the conflicts in the parser table, we first write the precedence rules

generally followed in regular expressions which is:

$$\text{Parentheses}(3e) > \text{Closure}(^*)(3d) > \text{Concatenation}(3c) > \text{Alternation}(|)(3b)$$

So, whenever there is a conflict (only shift-reduce conflicts in the table), we choose to continue with the action corresponding to the production rule having **higher precedence** in general. Other times, the choice is irrelevant, so we can choose any of the two actions. There is a straightforward correspondence between the production rules and the terms used, which has also been mentioned above. Now, we discuss the conflicts row-by-row:

- (6, (): The reduction rules would lead to the parser choosing concatenation. Shifting however, leads to parentheses. Since the parentheses define a completely new "scope", we can choose any of the two actions. We choose to **reduce** in this conflict.
- (9, (): The reduction rules would lead to the parser choosing concatenation. Shifting however, leads to parentheses. Since, there is implicit concatenation between the last seen R and the opening parenthesis, we can choose any of the two actions. We choose to **shift** in this case.
- (6, |): Shifting would lead to the alternation rule, while reducing would lead to the concatenation rule. Since concatenation takes higher precedence, we choose to **reduce** in this conflict.
- (9, |): Both actions lead to the alternation rule, therefore we can arbitrarily pick any action. We choose to **shift** in this conflict.
- (6, *): Shifting would lead to rule of closure, while reduction would lead to rule of concatenation. Since closure takes higher precedence, we choose to **shift** in this case.
- (9, *): Shifting would lead to rule of closure, while reduction would lead to rule of alternation. Since closure takes higher precedence, we choose to **shift** in this conflict.
- (6, a): Expressions produced by both shifting and reduction would eventually be concatenated. Thus, we arbitrarily choose to **shift**.
- (6, b): Analysis of this conflict is the same as above. Thus, we arbitrarily choose to **shift** in this conflict.
- (9, a): Shifting would lead to the last seen R being concatenated with the incoming a, while reduction would lead to alternation. Since concatenation takes higher precedence than alternation, we choose to **shift** in this conflict.
- (6, b): Analysis of this conflict is the same as above. Thus, we choose to **shift** in this conflict.

The updated parser table is provided below:

State	Action							GOTO
	(	|	)	*	a	b	$	R
0	s_2				s_3	s_4		1
1	s_2	s_5		s_7	s_3	s_4	*accept*	6
2	s_2				s_3	s_4		8
3	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	$r_{(3f)}$	
4	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	$r_{(3g)}$	
5	s_2				s_3	s_4		9
6	$r_{(3c)}$	$r_{(3c)}$	$r_{(3c)}$	s_7	s_3	s_4	$r_{(3c)}$	6
7	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	$r_{(3d)}$	
8	s_2	s_5	s_{10}	s_7	s_3	s_4		6
9	s_2	s_5	$r_{(3b)}$	s_7	s_3	s_4	$r_{(3b)}$	6
10	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	$r_{(3e)}$	

Problem 4

[50 marks]

The grammar has been written in the bison file `a2.y`. I used the commands `bison -d a2.y; flex a2.l; g++ a2.tab.c lex.yy.c -lfl`, but there was Segmentation Fault.