

EXAM

Please read all instructions, including these, carefully

- There are 7 questions on the exam, with multiple parts. You have 3 hours to work on the exam.
- The exam is open book, open notes.
- Please write your final answers in the space provided on the exam. You may use the backs of the exam pages as scratch paper, or use additional pages (available at the front of the room).
- Each problem has a straightforward solution. Solutions will be graded on correctness and clarity. Partial solutions will be given partial credit.

NAME : _____

Problem	Max points	Points
1	10	
2	10	
3	15	
4	10	
5	20	
6	20	
7	15	
TOTAL	100	

Eliminate left recursion:

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow |TE' \\ E' &\rightarrow \varepsilon \\ T &\rightarrow PT \\ T &\rightarrow \varepsilon \\ P &\rightarrow FP' \\ P' &\rightarrow *P' \\ P' &\rightarrow \varepsilon \\ F &\rightarrow (E) \\ F &\rightarrow a \\ F &\rightarrow b \end{aligned}$$

Nothing else needs to be done since the right hand sides of any given nonterminal have disjoint FIRST sets.

3. Consider the following simple context free grammar:

$$\begin{aligned} S &\rightarrow Aa \mid bAc \mid Bc \mid bBa \\ A &\rightarrow d \\ B &\rightarrow d \end{aligned}$$

a. Is this grammar $LR(1)$? Why or why not?

The grammar is not $LR(1)$. Consider the first few item sets:

$$\begin{aligned} l_0: & S \rightarrow \cdot Aa \\ & S \rightarrow \cdot bAc \\ & S \rightarrow \cdot Bc \\ & S \rightarrow \cdot bBa \\ & A \rightarrow \cdot d, a \\ & B \rightarrow \cdot d, c \\ l_1: & \text{GOTO}(l_0, A) \\ & S \rightarrow A \cdot a \\ l_2: & \text{GOTO}(l_0, b) \\ & S \rightarrow b \cdot Ac \\ & S \rightarrow b \cdot Ba \\ & A \rightarrow \cdot d, c \\ & B \rightarrow \cdot d, a \\ l_3: & \text{GOTO}(l_0, d) = \text{GOTO}(l_2, d) \\ & A \rightarrow d \cdot, a \\ & B \rightarrow d \cdot, c \\ & A \rightarrow d \cdot, c \\ & B \rightarrow d \cdot, a \end{aligned}$$

In state 3 we cannot decide whether to reduce to A or to B, since the lookaheads a and c are associated with productions for A and B.

b. Is it $LALR(1)$? Why or why not?

The simple answer is that since $LALR(1)$ grammars are a subset of $LR(1)$ grammars, if the grammar is not $LR(1)$ it cannot be $LALR(1)$.

You can see that the grammar is not $LALR(1)$ by collapsing productions with a common core in state 3, which does not resolve the ambiguity:

$$\begin{aligned} l_3: & \text{GOTO}(l_0, d) = \text{GOTO}(l_2, d) \\ & A \rightarrow d \cdot, \{a, c\} \\ & B \rightarrow d \cdot, \{a, c\} \end{aligned}$$

4. Consider the following grammar:

$$\begin{aligned}
 S &\rightarrow Bxxxxyz \\
 &\quad | Cxxxxz \\
 &\quad | xBxxxxxy \\
 B &\rightarrow w \\
 C &\rightarrow w
 \end{aligned}$$

For some values of i, j, k , the above grammar is $LR(i)$ and $SLR(j)$ but not $SLR(k)$.

Recall that $LR(i)$ will carry along i lookahead characters. The following is a fragment of the item sets:

$$\begin{aligned}
 I_0: & S \rightarrow \cdot Bxxxxyz \\
 & S \rightarrow \cdot Cxxxxz \\
 & S \rightarrow \cdot xBxxxxxy \\
 & B \rightarrow \cdot w, xxxy \\
 & C \rightarrow \cdot w, xxxx \\
 \\
 I_1: & \text{GOTO}(I_0, w) \\
 & B \rightarrow w \cdot, xxxy \\
 & C \rightarrow w \cdot, xxxx
 \end{aligned}$$

a. What is the smallest correct value for i ?

In state 1, you will need 4 characters lookahead to determine whether to reduce to B or C. Three characters lookahead would cause the entry in row (state) 1 and column (lookahead) xxx to indicate to reduce by both $B \rightarrow w$ and $C \rightarrow w$. So the smallest correct value for i is 4.

b. What is the smallest correct value for j ?

$SLR(j)$ uses follow sets for lookahead. So in state 1 above, we need to ensure that we cannot indicate a reduction by both $B \rightarrow w$ and $C \rightarrow w$ in the same cell of the parse table. If the lookahead were four characters as in part (a), there would be a conflict in the column for xxxx, since $FOLLOW(B)$ contains {xxxy, xxxx} and $FOLLOW(C)$ contains {xxxx}. One more lookahead character resolves the problem: now $FOLLOW(B)$ contains {xxxzy, xxxxy} and $FOLLOW(C)$ contains {xxxxz}. So the smallest correct value for j is 5.

c. What is the largest correct value for k ?

Following the explanation in part b, the largest correct value for k would be 4. With 5 character lookahead the SLR requirement is satisfied.

5. A context-free grammar over 0 and 1 has three non-terminals S, A , and B . We know the following about the FIRST and FOLLOW sets of these non-terminals:

$$\text{FIRST}(S) = \{0, 1\}$$

$$\text{FOLLOW}(S) = \{0, \$\}$$

$$\text{FIRST}(A) = \{\epsilon\}$$

$$\text{FOLLOW}(A) = \{0, 1\}$$

$$\text{FIRST}(B) = \{\epsilon\}$$

$$\text{FOLLOW}(B) = \{0, 1\}$$

- a. Construct a context-free grammar with these FIRST and FOLLOW sets.

There are several possible grammars; here is the most straightforward one:

Since $\text{FIRST}(S)$ contains 1 and 0, we can construct these productions:

$$S \rightarrow 0 \mid 1$$

$\text{FOLLOW}(S)$ contains \$ because it is the start symbol. To provide for $\text{FOLLOW}(S)$ containing 0 include the following:

$$S \rightarrow 0 \mid 1 \mid S0$$

A and B both include only the empty string in their FIRST sets, so we can include

$$A \rightarrow \epsilon$$

$$B \rightarrow \epsilon$$

Since $\text{FIRST}(S) = \text{FOLLOW}(A) = \text{FOLLOW}(B)$, we can modify the above to obtain the following final grammar:

$$S \rightarrow 0 \mid 1 \mid AS0 \mid BS0$$

$$A \rightarrow \epsilon$$

$$B \rightarrow \epsilon$$

- b. Construct the $LL(1)$ parsing table for your grammar. Is your grammar $LL(1)$?

	0	1	\$
S	$S \rightarrow 0$ $S \rightarrow AS0$ $S \rightarrow BS0$	$S \rightarrow 1$ $S \rightarrow AS0$ $S \rightarrow BS0$	accept
A	$A \rightarrow \epsilon$	$A \rightarrow \epsilon$	
B	$B \rightarrow \epsilon$	$B \rightarrow \epsilon$	

The grammar is clearly not $LL(1)$ since there are multiple entries in the table.

- c. Show the parsing steps for the string 10 with your $LL(1)$ parsing table, and trace a left-most derivation of this string.

Stack	Production applied
S	
AS0	$S \rightarrow AS0$
S0	$A \rightarrow \epsilon$
10	$S \rightarrow 1$

The leftmost derivation is $S \rightarrow AS0 \rightarrow S0 \rightarrow 10$.

6. Consider the following context-free grammar:

$$\begin{aligned} E' &\rightarrow E \\ E &\rightarrow E a T \mid T \\ T &\rightarrow T F \mid F \\ F &\rightarrow F b \mid c \end{aligned}$$

a. Construct the $LR(0)$ items and the DFA to recognize the viable prefixes for this grammar. Notice that the grammar is already augmented, and E' is the new start symbol.

$$\begin{aligned} l_0: & E' \rightarrow \bullet E \\ & E \rightarrow \bullet E a T \\ & E \rightarrow \bullet T \\ & T \rightarrow \bullet T F \\ & T \rightarrow \bullet F \\ & F \rightarrow \bullet F b \\ & F \rightarrow \bullet c \end{aligned}$$

$$\begin{aligned} l_1: & \text{GOTO}(l_0, E) \\ & E' \rightarrow E \bullet \\ & E \rightarrow E \bullet a T \end{aligned}$$

$$\begin{aligned} l_2: & \text{GOTO}(l_0, T) \\ & E \rightarrow T \bullet \\ & T \rightarrow T \bullet F \\ & F \rightarrow \bullet F b \\ & F \rightarrow \bullet c \end{aligned}$$

$$\begin{aligned} l_3: & \text{GOTO}(l_0, F) \\ & T \rightarrow F \bullet \\ & F \rightarrow F \bullet b \end{aligned}$$

$$\begin{aligned} l_4: & \text{GOTO}(l_0, c) = \text{GOTO}(l_2, c) = \text{GOTO}(l_8, c) = \\ & \text{GOTO}(l_8, c) \\ & F \rightarrow c \bullet \end{aligned}$$

$$\begin{aligned} l_5: & \text{GOTO}(l_1, a) \\ & E \rightarrow E a \bullet T \\ & T \rightarrow \bullet T F \\ & T \rightarrow \bullet F \\ & F \rightarrow \bullet F b \\ & F \rightarrow \bullet c \end{aligned}$$

$$\begin{aligned} l_6: & \text{GOTO}(l_2, F) = \text{GOTO}(l_8, F) \\ & T \rightarrow T F \bullet \\ & F \rightarrow F \bullet b \end{aligned}$$

$$\begin{aligned} l_7: & \text{GOTO}(l_3, b) = \text{GOTO}(l_6, b) \\ & F \rightarrow F b \bullet \end{aligned}$$

$$\begin{aligned} l_8: & \text{GOTO}(l_5, T) \\ & E \rightarrow E a T \bullet \\ & T \rightarrow T \bullet F \\ & F \rightarrow \bullet F b \\ & F \rightarrow \bullet c \end{aligned}$$

$$\begin{aligned} l_9: & \text{GOTO}(l_5, F) \\ & T \rightarrow T F \bullet \end{aligned}$$

$$\text{FIRST}(E) = \{c\}$$

$$\text{FIRST}(T) = \{c\}$$

$$\text{FIRST}(F) = \{c\}$$

$$\text{FOLLOW}(E) = \{a, \$\}$$

$$\text{FOLLOW}(T) = \{a, c, \$\}$$

$$\text{FOLLOW}(F) = \{a, b, c, \$\}$$

- b. Using the DFA that you constructed in part (a), build the *SLR(1)* parsing table for this grammar.

	ACTION				GO TO		
	a	b	c	\$	E	T	F
0			S4		1	2	3
1	S5			ACCEPT			
2	R E → T		S4				6
3	R T → F	S7	R T → F	R T → F			
4	R F → c	R F → c	R F → c	R F → c			
5	S8		S4			8	9
6	R T → TF	S7	R T → TF	R T → TF			
7	R F → Fb	R F → Fb	R F → Fb	R F → Fb			
8	R E → EaT			R E → EaT		9	
9	R T → F		R T → F	R T → F			6

- c. Show the parsing steps for the string *cbcac*.

Stack	Input	Action
[\$0]	cbcac\$	S4
[\$0][c4]	bcac\$	R F → c
[\$0][F3]	bcac\$	S7
[\$0][F3][b7]	cac\$	F → Fb
[\$0][F3]	cac\$	R T → F
[\$0][T2]	cac\$	S4
[\$0][T2][c4]	ac\$	R F → c
[\$0][T2][F6]	ac\$	R T → TF
[\$0][T2]	ac\$	R E → T
[\$0][E1]	ac\$	S5
[\$0][E1][a5]	c\$	S4
[\$0][E1][a5][c4]	\$	R F → c
[\$0][E1][a5][F9]	\$	R T → F
[\$0][E1][a5][T8]	\$	R E → EaT
[\$0][E1]	\$	accept

- d. Can you add a single additional production to the grammar to make it not *SLR(1)*? Give a brief explanation of your answer.

There are several productions that, if included in the grammar, would make it not *SLR(1)*. For example, if you added $F \rightarrow a$, there would be a shift-reduce conflict in states 2 and 8 since a is in $\text{FOLLOW}(T)$.

7. Consider the following context-free grammar that generates regular expressions.
- a. Define a syntax- directed transition that records the maximum number of *nested* kleene star operators of a regular expression R in its attribute $R.depth$. For example, the regular expression $(a)^* | ((b)^* | a)^*$ has depth 2. The semantic actions for the three base cases are given.

```
 $R \rightarrow$  a  
    {  $R.depth = 0;$  }  
| b  
    {  $R.depth = 0;$  }  
|  $\epsilon$   
    {  $R.depth = 0;$  }  
|  $R_1 R_2$   
  If  $R_1.depth > R_2.depth$   
     $R.depth = R_1.depth$   
  else  $R.depth = R_2.depth$   
|  $R_1 | R_2$   
  If  $R_1.depth > R_2.depth$   
     $R.depth = R_1.depth$   
  else  $R.depth = R_2.depth$   
|  $(R_1)$   
   $R.depth = R_1.depth$   
|  $(R_1)^*$   
   $R.depth = R_1.depth + 1$ 
```

- b. Is $R.depth$ inherited or synthesized? Explain your answer.

$R.depth$ is synthesized, because it is propagated up the parse tree starting at the leaves.

