

# FINAL EXAM

Please read all instructions (including these) carefully.

- There are 10 questions on the exam, some with multiple parts. You have a maximum of 3 hours to work on the exam.
- The exam is open book, open notes.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.

Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is more complicated than necessary. Partial solutions will be graded for partial credit.

NAME: \_\_\_\_\_

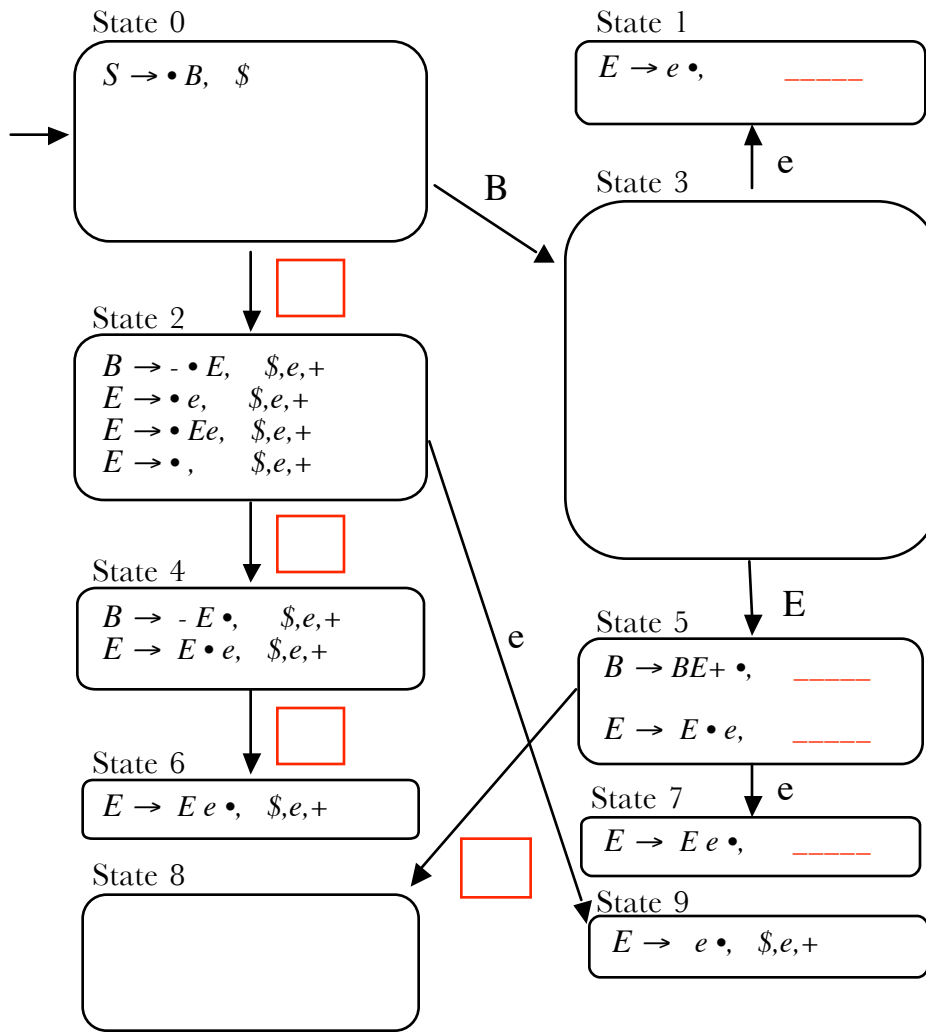
Problem	Max points	Points
1	20	
2	10	
3	10	
4	10	
5	10	
6	5	
7	5	
8	10	
9	10	
10	10	
TOTAL	100	

1. Consider the following grammar:

$$\begin{aligned}
 S &\rightarrow B \\
 B &\rightarrow BE+ \\
 &\quad | -E \\
 E &\rightarrow e \\
 &\quad | Ee \\
 &\quad | \varepsilon
 \end{aligned}$$

Below is a partial DFA for the grammar above. Notice we did not have to add a new start state (i.e., augment the grammar) as is usually done for LR(1) parsers.

- Complete state 0 by performing closure on the item listed. Don't forget to include lookahead symbols for the new items.
- Fill in all elements of states 3 and 8, and the lookahead items in states 1, 5 and 7.
- Fill in the missing transition labels (4 of them).



- (d) Fill in the following table with the reduce productions and lookahead tokens for the given states. Write “not a reduce state” if that state does not have a reduction action.

State	Production	Lookahead tokens
0		
2		
4		

- (e) For each state with a shift-reduce conflict, list the state, the lookahead token and the production that are in conflict. Write “none” if there are no shift-reduce conflicts in any state.

State	Production	Lookahead token
-------	------------	-----------------

- (f) For each state with a reduce-reduce conflict, list the state, and the conflicting productions. Write “none” if there are no reduce-reduce conflicts.

State	Conflicting productions
-------	-------------------------

- (g) For each of the following configurations of the LR(1) parser, say what action is taken by the parser (one of “shift”, “reduce with production ...”, “error” or “accept”) and write the next configuration (except in the case of an accept or error). You can assume that in the case of a shift-reduce conflict the parser chooses to shift. The  $\triangleright$  symbol indicates the current position in scanning the input string.

Configuration	Action	Next configuration
$- \triangleright e + \$$		
$BEe \triangleright e + \$$		
$Be \triangleright \$$		

2. Consider the following grammar in which  $S$  is the start non-terminal and the productions for the non-terminals  $A$ ,  $B$ , and  $C$  are not shown. There are no other productions for  $S$ .

$$\begin{aligned} S &\rightarrow AB \mid AC \\ A &\rightarrow \dots \\ B &\rightarrow \dots \\ C &\rightarrow \dots \end{aligned}$$

Consider now the LL(1) parsing table for this grammar. What conditions must be satisfied by the First and Follow sets of the non-terminal, such that the row corresponding to  $S$  in this table does not contain multiply-defined entries?

Write your answer in the table below considering six separate cases. For example, in the top left entry consider the case where  $\epsilon \notin \text{First}(B)$  and  $\epsilon \notin \text{First}(C)$  and  $\text{First}(A) = \{\epsilon\}$ . Write what **additional** conditions must be met, or write “never” if no additional conditions would help.

	$\epsilon \notin \text{First}(B)$ $\epsilon \notin \text{First}(C)$	$\epsilon \notin \text{First}(B)$ $\epsilon \in \text{First}(C)$	$\epsilon \in \text{First}(B)$ $\epsilon \in \text{First}(C)$
$\text{First}(A) = \{\epsilon\}$			
$\text{First}(A) \neq \{\epsilon\}$			

3. Consider the following grammar.

$$S \rightarrow S a \mid b \mid \mathbf{error} a$$

(a) Show the DFA for recognizing viable prefixes of this grammar. Use LR(0) items, and treat **error** as a terminal.

- (b) Some bottom-up parsers use error productions in the following way: When a parsing error is encountered (i.e., the machine cannot shift, reduce, or accept):
- a. First, pop and discard elements of the stack one at a time until a stack configuration is reached where the error terminal of an error production can be shifted on to the stack.
  - b. Second, discard tokens of the input one at a time until one is found that can be shifted on to the stack.
  - c. Third, resume normal execution of the parser.

Show the sequence of stack configurations of an SLR(1) parser for the grammar above on the following input. Be sure to show all shift, reduce, and discard actions (for both stack and input).

**bacadfa**

Stack (with top at right)	Input	Action

4. Consider the following grammar over the alphabet  $\{b, g, h, i\}$ :

$$\begin{aligned}
 A &\rightarrow B C D \\
 B &\rightarrow b B \mid \varepsilon \\
 C &\rightarrow C g \mid g \mid C h \mid i \\
 D &\rightarrow A B \mid \varepsilon
 \end{aligned}$$

(a) Fill in the table below with the First and Follow sets for the non-terminals in this grammar:

	<i>First</i>	<i>Follow</i>
A		
B		
C		
D		

(b) Fill in the column headings, and the row corresponding to B and C in the predictive LL(1) parsing table for this grammar. You may have some entries with more than one production

B				
C				

5. Write a translation scheme that computes the number of occurrences of the symbol  $a$  for any  $w \in G$ . Use only a single synthesized attribute, called *count*.

$$\begin{array}{l|l} 1 & S_1 \rightarrow aaS_2 \\ 2 & S_1 \rightarrow S_2bb \\ 3 & S_1 \rightarrow aS_2b \\ 4 & S \rightarrow a \\ 5 & S \rightarrow b \end{array}$$

If we eliminate rules 1 and 2 from the grammar  $G$  above, is the resulting grammar  $G'$  LL(1)? Justify your answer.

$$\begin{array}{l} S \rightarrow aSb \\ S \rightarrow a \\ S \rightarrow b \end{array}$$

6. Which of LL(1), SLR(1), and LR(1) can parse strings in the following grammar, and why? Justify your response.

$$\begin{array}{l} E \rightarrow A \mid B \\ A \rightarrow a \mid c \\ B \rightarrow b \mid c \end{array}$$

7. Consider the following grammar, in which **id**, **=**, **+**, and **string** are terminals:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow \text{id} = S \mid S + S \mid \text{id} \mid \text{string} \end{aligned}$$

Which of the following are viable prefixes of the language generated by the grammar? For each viable prefix, give a suffix that produces a right-sentential form.

- (a)  $S$
- (b)  $+ S +$
- (c)  $S + \text{id} =$

8. Consider the following grammar:

$$\begin{aligned} S &\rightarrow ScB \mid B \\ B &\rightarrow e \mid efg \mid efCg \\ C &\rightarrow SdC \mid S \end{aligned}$$

Give an LL(1) grammar that generates the same language.

9. Give a grammar with the following First and Follow sets. Your grammar should have two productions per non-terminal and no epsilon productions. The non-terminals are  $X, Y, Z$  and the terminals are  $a, b, c, d, e, f$ .

$$\begin{array}{l|l}
 \text{First}(X) = \{b, d, f\} & \text{Follow}(X) = \{\$\} \\
 \text{First}(Y) = \{b, d\} & \text{Follow}(Y) = \{c, e\} \\
 \text{First}(Z) = \{c, e\} & \text{Follow}(Z) = \{a\} \\
 \text{Follow}(d) = \{c, e\} & \text{Follow}(a) = \{\$\} \\
 \text{Follow}(e) = \{a\} & \text{Follow}(b) = \{b, d\} \\
 \text{Follow}(f) = \{\$\} & \text{Follow}(c) = \{c, e\}
 \end{array}$$

There are many ways to solve this problem. In our solution, we began by using the First sets to generate the obvious productions that have the correct first (and only) terminal:

$$\begin{array}{l}
 X \rightarrow b \mid d \mid f \\
 Y \rightarrow b \mid d \\
 Z \rightarrow c \mid e
 \end{array}$$

We are only allowed two productions per nonterminal, so we need to eliminate an X production. Noticing that b and d are in the first of Y, we can try

$$\begin{array}{l}
 X \rightarrow Y \mid f \\
 Y \rightarrow b \mid d \\
 Z \rightarrow c \mid e
 \end{array}$$

At this point we have the First sets correct for the nonterminals; now we try to modify the productions to have the correct Follow sets as well. Starting arbitrarily with the terminal c, we notice that both c and e are in Follow(Z) and in First(Z), so we can add Z after c in some production. There is only one choice:

$$\begin{array}{l}
 X \rightarrow Y \mid f \\
 Y \rightarrow b \mid d \\
 Z \rightarrow cZ \mid e
 \end{array}$$

Similar reasoning about Follow(b) leads to:

$$\begin{array}{l}
 X \rightarrow Y \mid f \\
 Y \rightarrow bY \mid d \\
 Z \rightarrow cZ \mid e
 \end{array}$$

Since \$ is not in the follow of Y, there must be something that comes after Y in the  $X \rightarrow Y$  production.

Since First(Z) is in the Follow(Y) we try:

$$\begin{array}{l}
 X \rightarrow YZ \mid f \\
 Y \rightarrow bY \mid d \\
 Z \rightarrow cZ \mid e
 \end{array}$$

At this point we still haven't used a. Now Follow(a) = {\$} and Follow(Z) = {a}. So we get

$$\begin{array}{l}
 X \rightarrow YZa \mid f \\
 Y \rightarrow bY \mid d \\
 Z \rightarrow cZ \mid e
 \end{array}$$

At this point it is routine to check that all the requirements are satisfied.

10. Consider the syntax-directed definition below, in which *mknnode* constructs internal syntax tree nodes with two children; *mkleaf* constructs leaf syntax tree nodes, and *nptr* for *E* and *T* keeps track of the pointers returned by the function calls to *mknnode* and *mkleaf*.

$E \rightarrow E_1 + T$	$E.nptr = mknnode(+, E_1.nptr, T.nptr)$
$E \rightarrow T$	$E.nptr = T.nptr$
$T \rightarrow ( E_1 )$	$T.nptr = E_1.nptr$
$T \rightarrow id$	$T.nptr := mkleaf(id, id.lexeme)$
$T \rightarrow num$	$T.nptr := mkleaf(num, num.val)$

- (a) Construct the parse tree and syntax tree for the expression **((a)+(b))**.

- (b) Say whether each attribute of a non-terminal is *inherited* or *synthesized* and why.