CS331 - Compilers

Spring, 2001

NAME:_____

ANSWER ALL QUESTIONS IN THE BLUE BOOKS PROVIDED. AFTER THE EXAM RETURN BOTH THE BLUE BOOK AND YOUR COPY OF THE EXAM. IF YOU ARE UNSURE ABOUT THE MEANING OF A QUESTION, INDICATE THE ASSUMPTIONS YOU MAKE IN ANSWERING IT. APPROPRIATE CREDIT WILL BE GIVEN FOR REASONABLE ASSUMPTIONS. POINT VALUES FOR EACH QUESTION ARE GIVEN IN PARENTHESES.

Note: in all example grammars, the terminal symbols are given in boldface.

1. (10)

a. Consider the grammar

S → S**a** | **b**

Show that FIRST sets are unable to guide a top-down parser in selecting the correct right-hand side when parsing a string such as *baaaaa*.

b. Remove immediate left recursion from the grammar

A → Ab | cA | Bd | e

2. (10)

a. Construct FIRST and FOLLOW sets for the following grammar.

S → A**bb** | C
A → **a**A | **b**
C → **ab** | **cde**

b. Indicate whether or not the grammar is LL(1), and explain why or why not.

3. (10) Construct a *non-deterministic, bottom-up* parser for the following grammar:

E → E **+** E | E ***** E | **a** | **(** E **)**

4. (10) Consider the grammar

S → E
E → E **+** T | T
T → T ***** F | F
F → E | **d**

The FOLLOW sets for this grammar are

FOLLOW(S) = { $ }
FOLLOW(E) = { $) + }
FOLLOW(T) = { $) + * }
FOLLOW(F) = { $) + * }

a. Is the grammar LL(1)? If not, why not?

b. Is the grammar LR(0) (i.e., parseable by an LR(0) parser)? If not, why not?

c. Is the grammar SLR(1) (i.e., parseable by an SLR(1) parser)? If not, why not?

5. (10) The construction of an SLR(1) parsing table requires collecting sets of LR(0) items, which group viable prefixes of the grammar into state sets that are later used define the SLR(1) table entries.

Suppose that one of the state sets (call it I8) in the canonical collection of LR(0) items for some grammar consists of the following items:

E → T •
T → T • * F

(NOTE: '•' is the dot in LR items.)

a. What entry (or entries) in the parse table would be made on the basis of the item

E → T •

(Note: FOLLOW(E) = {), +, $})

b. What entry or entries will be made on the basis of the item T → T • * F (assume another set of items (call it I9) includes the item T → T * • F)?

c. What problem would arise if FOLLOW(E) contained '*'?

6. (10) LR parsers incorporate into their mechanism a finite state automaton (FSA) for recognizing viable prefixes of the grammar. Given the table below, reconstruct the FSA that defines the table entries. (NOTE: construct the FSA *only*--with states and transitions on terminal and non-terminal symbols. Do not include items from the canaonical collection or production rules.)

	ACTION						GO TO		
	a	b	c	d	e	$	S	A	C
0		S3			S4		1	2	
1						ACCEPT			
2			S5						
3				S6		R, S -> b			
4	S8		S9						7
5						R, S -> Ac			
6	S8		S9						10
7			R, A -> eC						
8				S11					
9	S8		S9						12
10			R, A -> bdC						
11			R, C -> aD						
12			R, C -> cC						

7. (10) Consider the grammar

(1) N → NB
(2) N → ε
(3) B → **1**
(4) B → **0**

Show a bottom-up parse of the string **1101**, using the LR parse table below:

State	1	0	$	N	B
0	s3	s4		1	2
1	s3	s4	acc		5
2	r2	r2	r2		
3	r3	r3	r3		
4	r4	r4	r4		
5	r1	r1	r1		

NOTE: productions are numbered as in the grammar above.

8. (5) Can there ever be symbols between *A* and *a* in a sentential form ...*A*...*a*... for *a* in FOLLOW(A)?

9. (10) Given the grammar below:

Z → S
S → **w**S
S → AB
A → **x**A
A → **y**
B → **z**

a. Show a top-down parse (give the stack, input remaining, and production applied). The initial configuration is given here:

Stack	**Input**	**Production**
$Z	wwxyz$	

b. Draw the parse tree for the parse you just outlined.

10. (10)

a. Show that the grammar below is ambiguous.

b. Change it to be unambiguous, but still generate the same set of strings.

number → digit | number number
digit → **0** | **1** | **2** |. . . | **9**

11. (5) Given the grammar below, describe the entries that will be made in the parse table in the rows labeled E and E' (give the label of the column as well as the cell contents). Note: the question itself should tell you whether the parse table is for a bottom-up or top-down parser, so don't ask!

E → T E'
E' → **+** T E' | ε
T → F T'
T' → ***** F T' | ε
F → **Id** | (E)