CS331 Compiler Design
Spring 2005

EXAM

Please read all instructions, including these, carefully

- There are 9 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 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, 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.

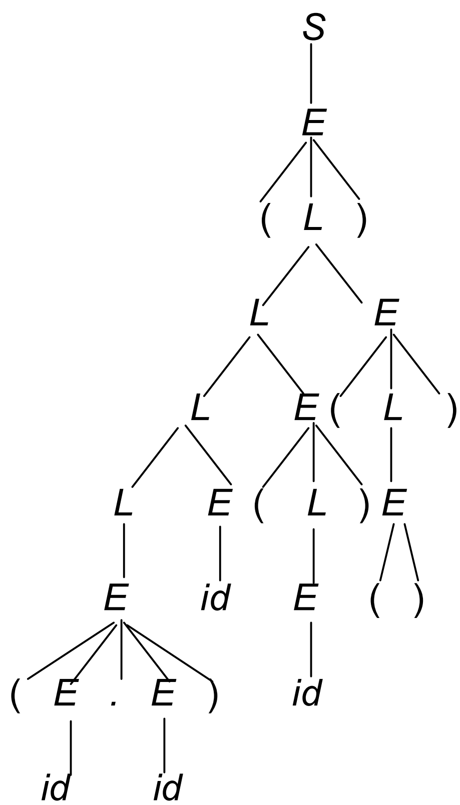
NAME : ____

Problem	Max points	Points
1	10	
2	15	
3	10	
4	10	
5	10	
6	10	
7	10	
8	10	
9	15	
TOTAL	100	

1. (a) Using the grammar below, draw a derivation tree for the following string:

((id . id) id (id) (( )))

$$
\begin{array}{lll}
S & \rightarrow & E \\
E & \rightarrow & \text{id} \\
 & & \mid (\ E\ .\ E\ ) \\
 & & \mid (\ L\ ) \\
 & & \mid (\ ) \\
L & \rightarrow & L\ E \\
 & & \mid E
\end{array}
$$



(b) Give a rightmost canonical derivation for the string given in part (a).

S

E

(L)

(L E)

(L(L))

(L (E))

(L (( )))

(L E (( )))

(L (L) (( )))

(L (E) (( )))

(L (id) (( )))

(L E (id) (( )))

(L id (id) (( )))

(E id (id) (( )))

((E . E) id (id) (( )))

((E . id) id (id) (( )))

((id . id) id (id) (( )))

2. Consider the following grammar:

1 $S \rightarrow A\ B\ A$
2 $A \rightarrow B\ c$
3 $\quad\ \mid d\ A$
4 $\quad\ \mid \varepsilon$
5 $B \rightarrow eA$

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

	First	*Follow*
S	*d, e*	*$*
A	*d, e, ε*	*c, d, e, $*
B	*e*	*c, d, e*

(b) Fill in the LL (1) parse table for this grammar.

	c	*d*	*e*	*$*
S		1	1	
A	4	3,4	2,4	4
B			5	

3. Consider the following grammar for boolean expressions (words in capitals are terminals).

$E \rightarrow E$ OR E
$E \rightarrow E$ AND E
$E \rightarrow$ NOT E
$E \rightarrow (E)$
$E \rightarrow$ TRUE
$E \rightarrow$ FALSE
$E \rightarrow$ ID

(a) Show that this grammar is ambiguous.

There are two parse trees for several strings generated by this grammar. For example, the string *ID AND ID OR ID* has two parse trees:

(b) Rewrite the grammar to remove the ambiguity and enforce the intended precedence order by introducing new non-terminals. Make sure that your revised grammar accepts the same language as the original.

E → NOT E
  | E'
E' → E' OR T
  | T
T → T AND F
  | F
F → TRUE
  | FALSE
  | ID
  | (E)

4. Consider the following context free grammar:

$$S \rightarrow Aa$$
$$\mid bAc$$
$$\mid dc$$
$$\mid bda$$
$$A \rightarrow d$$

(a) Is the grammar SLR(1)? Why or why not? You don't need to construct the parse table to answer.

No, because when the stack contains *bd* with *a* next in input, there is a shift-reduce conflict. That is, we reach a certain state in the DFA by shifting *b* and then *d*. Since *a* is in FOLLOW(*A*), reduce by $A \rightarrow d$ will be indicated in the parse table cell indexed by the current state and the terminal *a.* However, because of the production $S \rightarrow bda$, shift *a* will also be indicated in the same cell.

The relevant item sets:

shift *b* ⇒
$S \rightarrow b{\bullet}Ac$
$S \rightarrow b{\bullet}da$
$A \rightarrow {\bullet}d$

shift *d* ⇒
$S \rightarrow bd{\bullet}a$ ⇐ shift put in cell for this state in column *a*
$A \rightarrow d{\bullet}$ ⇐ reduce put in cell for this state in columns for all terminals in FOLLOW(A) = { *a, c* }

(b) Is the grammar LR(1)? Why or why not?

Yes, because the explicit lookahead will prevent the reduce move from being indicated in the cell.

The relevant item sets:

shift *b* ⇒
$S \rightarrow b{\bullet}Ac$
$S \rightarrow b{\bullet}da$
$A \rightarrow {\bullet}d$, c

shift *d* ⇒
$S \rightarrow bA{\bullet}c$
$S \rightarrow bd{\bullet}a$ ⇐ shift put in cell for this state in column *a*
$A \rightarrow d{\bullet}$, c ⇐ reduce put in cell for this state in column *c* only

(c) Consider the following context free grammar for the same language:

$$S \rightarrow Aa$$
$$\mid bAc$$
$$\mid Bc$$
$$\mid bBa$$
$$A \rightarrow d$$
$$B \rightarrow d$$

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

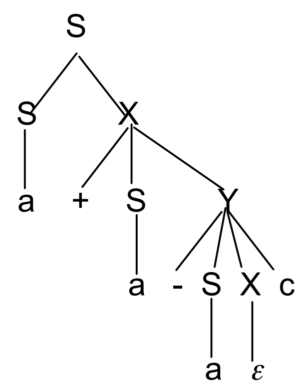
Yes. After shifting *b* and *d*, lookahead will ensure that we reduce to *A* if a *c* is in the input, and reduce to *B* if an *a* is in the input.

5. Let G be the following grammar:

1	$S \rightarrow S\,X$
2	$\mid\ a$
3	$X \rightarrow \varepsilon$
4	$\mid\ +\,S\,Y$
5	$\mid\ Y\,b$
6	$Y \rightarrow \varepsilon$
7	$\mid$ - S X c

(a) Show the parse tree for the following string :

a + a - ac



(b) Complete the table below tracing an LR(1) parse of the same string. The *Stack* column shows the stack (with the top at right), the *Input* column shows the not-yet-processed input, and the *Action* column shows whether the parser performs a shift action, a reduce action, or accepts the input. In the case of a reduce action, indicate (by number) which production is used. (NOTE: you do not need to construct the parse table; just use your knowledge of how the parser works and the parse tree above).

Stack (with top at right)	Input	Action
$	a+a-ac$	shift
$a	+a-ac$	reduce
$S	+a-ac$	shift
$S+	a-ac$	shift
$S+a	-ac$	reduce
$S+S	-ac$	shift
$S+S-	ac$	shift
$S+S-a	c$	reduce
$S+S-S	c$	reduce
$S+S-SX	c$	shift
$S+S-SXc	$	reduce
$S+SY	$	reduce
$SX	$	reduce
$S	$	accept

6. You may give any context-free grammar in answer to the questions in this part; i.e it doesn't matter if the grammar is LL (1), unambiguous, or any other subclass of all context free grammars.

(a) Consider the following FIRST and FOLLOW sets:

$$
\begin{aligned}
\text{FIRST}(S) &= \{b, \varepsilon\} \\
\text{FIRST}(T) &= \{b, \varepsilon\} \\
\text{FOLLOW}(S) &= \{a, \$\} \\
\text{FOLLOW}(T) &= \{a, b, \$\}
\end{aligned}
$$

Give the simplest grammar (fewest productions and shortest right-hand sides) that produces these sets. *S* is the start symbol.

There are several possible correct answers to this question. Here is one of them:

$S \rightarrow Tb \mid T$
$T \rightarrow bSa \mid \varepsilon$

(b) Give a grammar with a single non-terminal *S* such that FIRST(*S*) = ∅.

S → S
A common error is S → ε, which has ε in FIRST(S)

(c) Give a grammar with a single non-terminal *S* such that FIRST(*S*) = (FOLLOW(*S*) − {\$}) = ∅.

S → a*S*a
Common errors include having FIRST(*S*) empty or FOLLOW(*S*) empty.

(d) Give a simple example of a grammar with a shift-reduce conflict.

$S \rightarrow Ta \mid a$
$T \rightarrow \varepsilon$

7. For each grammar explain why, or why not, the grammar is LL (1):

(a) $S \rightarrow 0 \mid 12 \mid 345$

Clearly LL(1), as each production rhs begins with a distinct terminal.

(b) $S \rightarrow 0 \mid T1$
$T \rightarrow 1 \mid S0$

Not LL(1) because it is left-recursive: $S \Rightarrow T1 \Rightarrow S01$

(c) $S \rightarrow 0 \mid 11 \mid 01$

Not LL(1), because two of *S*'s right hand sides begin with the same terminal.

8. Let synthesized attribute *F.val* give the value of the binary fraction generated by *F* in the grammar that follows:

$$F \rightarrow \mathbf{.}\ L$$
$$L \rightarrow LB \mid B$$
$$B \rightarrow 0 \mid 1$$

For instance, on input **.101**, *F.val* = .625.

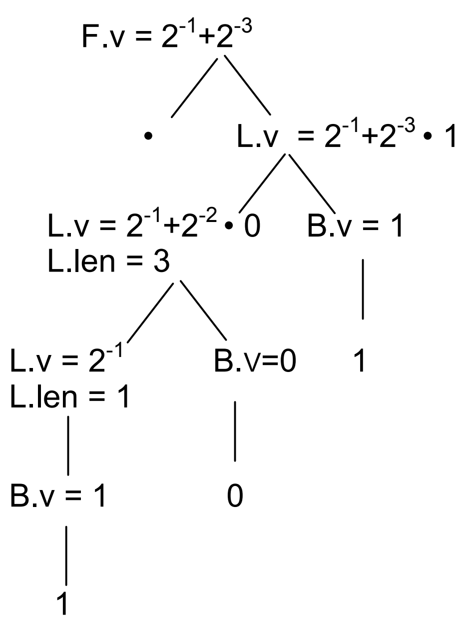
(a) Using only synthesized attributes, give a translation scheme for the above grammar.

You need two attributes; one to hold the current value (.v) and one to hold the length of the string (.len). There are several ways to do it:

	Answer 1	Answer 2	Answer 3
$F \rightarrow .L$	{F.v=L.v}	{F.v=L.v $\cdot\ 2^{-L.\text{len}}$ }	{F.v=L.v/2}
$L \rightarrow L_1 B$	{L.len= L_1.len+1 , L.v= L_1.v+$2^{-L.\text{len}}$ $\cdot$ B.v)}	{L.len= L_1.len+1, L.v= L_1.v $\cdot$ 2+B.v}	{L.len= L_1.len+1, L.v = L_1.v+$2^{-L.\text{len}}$ $\cdot$ B.v}
$L \rightarrow B$	{L.len=1 , L.v=B.v/2}	{L.len=1, L.v=B.v}	{L.len=1, L.v=B.v}
$B \rightarrow 0$	{B.v=0}	{B.v=0}	{B.v=0}
$B \rightarrow 1$	{B.v=1}	{B.v=1}	{B.v=1}

(b) Show the translation of the input string **.101** by decorating its parse tree. Make sure your tree is drawn clearly.

Here is the tree for Answer 1:



9. Consider the following parsing automaton. All that is shown are the states, transitions, and the *Left-hand side* of each item that has the "•" *all the way to right*. Complete the partial items and fill in all of the missing items (be sure to include the "•"). Note that the question requires only LR(0) items.