

FINAL EXAM

Please read all instructions (including these) carefully.

- There are 8 questions on the exam, 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, 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 graded for partial credit.

NAME: _____

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

(1) Consider the following grammar:

$$\begin{aligned} A' &\rightarrow A \\ A &\rightarrow xA \mid yA \mid y \end{aligned}$$

$$\mathbf{FIRST}(A) = \{x, y\}$$

$$\mathbf{FIRST}(xA) = \{x\}$$

$$\mathbf{FIRST}(yA) = \{y\}$$

$$\mathbf{FIRST}(y) = \{y\}$$

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

	x	y	$\$$
A'	$A' \rightarrow A$	$A' \rightarrow A$	
A	$A \rightarrow xA$	$A \rightarrow yA$ $A \rightarrow y$	

(a) Explain why this grammar is not LL(1).

$A \rightarrow yA$ and $A \rightarrow y$, so there is a conflict in the table in entry $[A, y]$

(a) Transform the grammar into a grammar that is LL(1).

Use left factoring to produce the following grammar:

$$\begin{aligned} A' &\rightarrow A \\ A &\rightarrow xA \mid yB \\ B &\rightarrow A \mid \epsilon \end{aligned}$$

(a) Give the parse table for the grammar created in (c). If you created additional non-terminals in your transformation, you will have to add rows to the table.

	x	y	$\$$
A'	$A' \rightarrow A$	$A' \rightarrow A$	
A	$A \rightarrow xA$	$A \rightarrow yB$	
B	$B \rightarrow A$	$B \rightarrow A$	$B \rightarrow \epsilon$

(2) In this question you will determine if the (original) grammar in the previous question is SLR(1). To answer, construct the following:

(a) The canonical collection of LR(0) items

$$S_0 = \{ [A' \rightarrow \bullet A], [A \rightarrow \bullet xA], [A \rightarrow \bullet yA], [A \rightarrow \bullet y] \}$$

$$S_1 = \text{GOTO}(S_0, A) = \{ [A' \rightarrow A \bullet] \}$$

$$S_2 = \text{GOTO}(S_0, y) = \{ [A \rightarrow y \bullet A], [A \rightarrow y \bullet], [A \rightarrow \bullet xA], [A \rightarrow \bullet yA], [A \rightarrow \bullet y] \}$$

$$S_3 = \text{GOTO}(S_0, x) = \{ [A \rightarrow x \bullet A], [A \rightarrow \bullet xA], [A \rightarrow \bullet yA], [A \rightarrow \bullet y] \}$$

$$S_4 = \text{GOTO}(S_2, A) = \{ [A \rightarrow yA \bullet] \}$$

$$S_5 = \text{GOTO}(S_3, A) = \{ [A \rightarrow xA \bullet] \}$$

$$\text{GOTO}(S_2, y) = \text{GOTO}(S_3, y) = S_2$$

$$\text{GOTO}(S_3, x) = \text{GOTO}(S_2, x) = S_3$$

Also consider: FOLLOW(A) = FOLLOW(A') = {\$}

(a) The SLR(1) parse table

	<i>x</i>	<i>y</i>	\$	A'	A
0	Shift 3	Shift 2			1
1			accept		
2	Shift 3	Shift 2	reduce A → y		4
3					5
4			reduce A → yA		
5			reduce A → xA		

(a) Is the grammar SLR(1)? Explain your answer.

Yes it is SLR(1), since there are no conflicts in the parse table.

(3) For the same grammar as in the previous two questions, construct the following:

(a) The canonical collection of LR(1) items

Same as for the SLR(1) grammar above, except lookaheads are added:

$$S_0 = \{ [A' \rightarrow A, \$], [A \rightarrow \bullet xA, \$], [A \rightarrow yA, \$], [A \rightarrow \bullet y, \$] \}$$

$$S_1 = \text{GOTO}(S_0, A) = \{ [A' \rightarrow A\bullet, \$] \}$$

$$S_2 = \text{GOTO}(S_0, y) = \{ [A \rightarrow y\bullet A, \$], [A \rightarrow y\bullet, \$], [A \rightarrow xA, \$], \\ [A \rightarrow \bullet yA, \$], [A \rightarrow y, \$] \}$$

$$S_3 = \text{GOTO}(S_0, x) = \{ [A \rightarrow x\bullet A, \$], [A \rightarrow \bullet xA, \$], \\ [A \rightarrow \bullet yA, \$], [A \rightarrow \bullet y, \$] \}$$

$$S_4 = \text{GOTO}(S_2, A) = \{ [A \rightarrow yA\bullet, \$] \}$$

$$S_5 = \text{GOTO}(S_3, A) = \{ [A \rightarrow xA\bullet, \$] \}$$

$$\text{GOTO}(S_2, y) = \text{GOTO}(S_3, y) = S_2$$

$$\text{GOTO}(S_3, x) = \text{GOTO}(S_2, x) = S_3$$

(a) The LR(1) parse table

	x	y	$\$$	A'	A
0	Shift 3	Shift 2			1
1			accept		
2	Shift 3	Shift 2	reduce $A \rightarrow y$		4
3					5
4			reduce $A \rightarrow yA$		
5			reduce $A \rightarrow xA$		

(a) Is the grammar LR(1)? Explain your answer.

There is nothing changed from the SLR(1) parse table, since $\$$ is the only symbol in FOLLOW(A). In any case, if a grammar is SLR(1) then it is certainly LR(1).

- (4) (a) Give an *unambiguous* grammar that is not LR(1).

There is an infinite number of unambiguous non-LR(1) grammars. Here is one of the simpler ones:

$$\begin{aligned} S &\rightarrow Axy \mid Bxz \\ A &\rightarrow a \\ B &\rightarrow a \end{aligned}$$

The grammar is unambiguous; it contains exactly two strings, each of which has a unique derivation:

However, the grammar is not LR(1). Consider an input that begins with ax . After shifting a , the LR(1) parser must decide whether to reduce a to A or B . Based solely on the lookahead character x , we cannot decide which production to choose; only by looking two characters ahead to see if a y appears (in which case, reduce to A) or a z appears (in which case, reduce to a B) can the decision be deterministically resolved.

- (b) Explain why the following grammar is LL(1) but not SLR(1):

$$\begin{aligned} X &\rightarrow YaYb \mid \tilde{z}b\tilde{z}a \\ Y &\rightarrow \varepsilon \\ \tilde{z} &\rightarrow \varepsilon \end{aligned}$$

The grammar is LL(1) because one of the right hand sides of X has $\{a\}$ as its FIRST set, and the other has $\{b\}$, and the right hand sides of Y and Z are unique. This is the only requirement for an LL(1) grammar. The grammar is not SLR(1), though, because SLR(1) parsers use FOLLOW sets to determine the lookahead symbols. When the parse table is built, the first set of items will contain

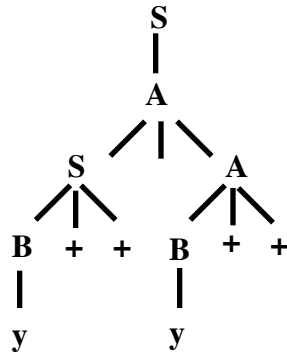
$$\begin{aligned} X &\rightarrow \bullet YaYb \\ X &\rightarrow \bullet ZbZa \\ Y &\rightarrow \bullet \\ Z &\rightarrow \bullet \end{aligned}$$

Since both a and b are in the FOLLOW sets for Y and Z , the reductions to both $Y \rightarrow \varepsilon$ and $Z \rightarrow \varepsilon$ will be entered in the columns headed by a and b , thus producing a REDUCE-REDUCE conflict. So, the grammar is not SLR(1).

(5) Consider the following grammar.

$$\begin{aligned}
 S &\rightarrow A \\
 A &\rightarrow A+A \mid B++ \\
 B &\rightarrow y
 \end{aligned}$$

(a) Draw the parse tree for the input “y + + + y + +”



(a) Complete the table below tracing an LR(1) parse of the input above. 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 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 given in the previous part.)

Stack (with top at right)	Input	Action
\$	y+++y++\$	shift
\$y	+++y++\$	reduce $B \rightarrow y$
\$B	+++y++\$	shift
\$B+	++y++\$	shift
\$B++	+y++\$	reduce $A \rightarrow B + +$
\$A	+y++\$	shift
\$A+	y++\$	shift
\$A+y	++\$	reduce $B \rightarrow y$
\$A+B	++\$	shift
\$A+B+	+\$	shift
\$A+B++	\$	reduce $A \rightarrow B + +$
\$A+A	\$	reduce $A \rightarrow A + A$
\$A	\$	reduce $S \rightarrow A$
\$S	\$	accept

(6) Consider the following grammar :

- 1 $S \rightarrow [SX]$
- 2 $\mid a$
- 3 $X \rightarrow \epsilon$
- 4 $\mid +SY$
- 5 $\mid Yb$
- 6 $Y \rightarrow \epsilon$
- 7 $\mid -SXC$

(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>
S	a, [\$,+,-,],c,b
X	+, -, b, ϵ], c
Y	-, ϵ], c, b

(b) Using the parse table for the grammar above, show a top-down parse of the string $[a+a-ac]$. The numbers in the cells refer to productions, as numbered in the grammar.

	a	b	c	+	-	[]	\$
S	2					1		
X		5	3	4	5		3	
Y		6	6		7		6	

Stack (with top at left)	Input	Action
S \$	[a+a-ac]\$	Pop S; push [S X]
[S X] \$	[a+a-ac]\$	Match [
S X] \$	a+a-ac]\$	Pop S; push a
a X] \$	a+a-ac]\$	Match a
X] \$	+a-ac]\$	Pop X; push + S Y
+ S Y] \$	+a-ac]\$	Match +
S Y] \$	a-ac]\$	Pop S; push a
a Y] \$	a-ac]\$	Match a
Y] \$	-ac]\$	Pop Y; push - S X c
- S X c] \$	-ac]\$	Match -
S X c] \$	ac]\$	Pop S; push a
a X c] \$	c]\$	Match a
X c] \$	c]\$	Pop X; push ϵ
c] \$	c]\$	Match c
] \$] \$	Match]
\$	\$	ACCEPT

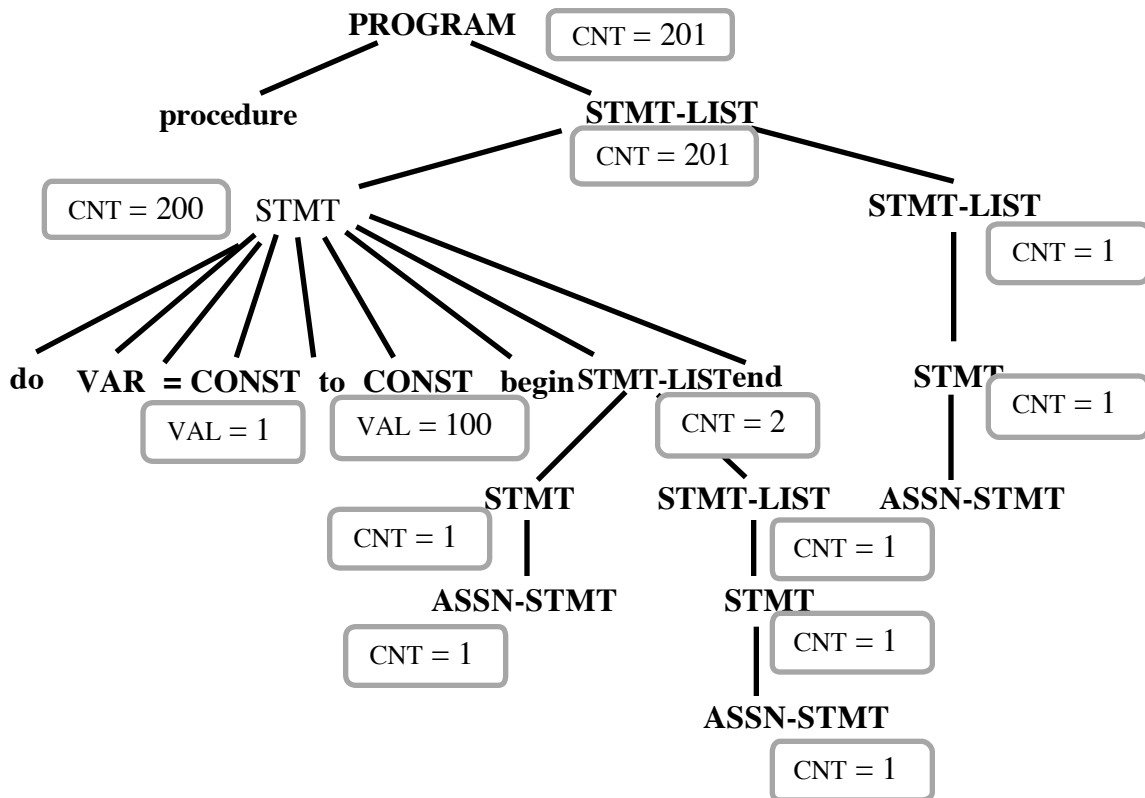
- (7) The following context-free grammar describes part of the syntax of a Pascal-like language. Non-terminals are given in capitals and terminals in lower case. VAR represents a variable name and CONST represents a constant. The productions for ASSN-STMT are not given, as they do not affect the question.

- 1 PROGRAM → procedure STMT-LIST
- 2 STMT-LIST → STMT STMT-LIST
- 3 | STMT
- 4 STMT → do VAR = CONST to CONST begin STMT-LIST end
- 5 | ASSN-STMT

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

```

procedure
  do i = 1 to 100 begin
    ASSN-STMT
    ASSN-STMT
  end
  ASSN-STMT
  
```



Write an attribute grammar that computes the number of executed statements for a program conforming to this grammar. Use the fact that the number of times a loop executes can be computed from the two constants that specify the range of the loop variable. You may assume that the lower bound of the range is less than or equal to the upper bound.

PROGRAM \rightarrow procedure STMT-LIST	PROGRAM.cnt = STMT LIST.cnt
STMT-LIST ₀ \rightarrow STMT STMT-LIST ₁	STMT LIST0.cnt = STMT.cnt + STMT LIST1.cnt
STMT-LIST \rightarrow STMT	STMT LIST.cnt = STMT.cnt
STMT \rightarrow do VAR = CONST ₁ to CONST ₂ begin STMT-LIST end	STMT.cnt = STMT LIST.cnt * (CONST2.val _ CONST1.val) + 1)
STMT \rightarrow ASSN-STMT	STMT.cnt = 1

- (b) Annotate the parse tree for the program fragment given in part (a) with the computed attribute values. (Annotate the parse tree you drew on the previous page, rather than re-draw it here.)
- (c) For each attribute used in your attribute grammar, indicate whether it is *synthesized* or *inherited*.

Both CNT and VAL are synthesized attributes because their values are propagated up the tree.

(8) Consider the following grammar:

$$\begin{aligned} S &\rightarrow aS \mid Ab \\ A &\rightarrow XY\zeta \mid \varepsilon \\ X &\rightarrow cS \mid \varepsilon \\ Y &\rightarrow dS \mid \varepsilon \\ \zeta &\rightarrow eS \end{aligned}$$

(a) Give a leftmost derivation of the string *aebb*.

S
aS
aAb
aXYZb
aYZb
aZb
aeSb
aeAbb
aebb

(b) Explain why it is that if we add the production $X \rightarrow bS$, the grammar is no longer LL(1).

If we add $X \rightarrow bS$, then the FIRST sets for A's two right hand sides become non-disjoint. That is, the FIRST set for the right hand side of $A \rightarrow XYZ$ will contain FIRST(X), which now includes *b*, but $A \rightarrow \varepsilon$ will also contain *b* because it is in FOLLOW(A).