CS 502 – Compiling and Programming Systems
Mid-term Examination, 3/24/11

Instructions: Read carefully through the whole exam first and plan your time. Note the relative weight of each question and part (as a percentage of the score for the whole exam). The total points is 100 (*ie*, your grade will be the percentage of your answers that are correct).

This exam is **open book, open notes**. You are free to refer to any book or other study materials you bring to the exam room.

You have **75 minutes** to complete all five (5) questions. Write your answers on this paper (use both sides if necessary).

Name:

Student Number:

Signature:

1. (10%) This is a thought/essay question (but don't write *too* much!). In FORTRAN FORMAT statements, there can appear a lexical entity known as a *Hollerith constant*. Its form is some decimal digits giving a count (say *n*), the letter `H`, and then *n* characters. The overall effect is to form a string constant *n* characters long. Examples include `3HYES`, `2HNO`, and `5HX␣Y␣Z`, where ␣ indicates a space character.

   (a) (5%) Is it possible to specify the language of Hollerith constants using an NFA or DFA? Is the language of Hollerith constants *regular*?
   Answer:

   The language of Hollerith constants cannot be specified using an NFA/DFA; it is *not* regular. The counting involved in counting out the number of characters to the right of the H requires an unbounded number of states.

   (b) (5%) Would Hollerith constants be easy or hard to process with a table driven scanner? With a hand-written one? Explain your easy/hard judgments.
   Answer:

   Hollerith constants would be difficult or impossible to handle with a pure table driven scanner, because they are not regular. If one could extend the table driven scanner to (a) associate arbitrary actions with any transition and (b) allow actions to help decide the target state of a transition, then one could process Hollerith constants with such an extended table driven scanner. One would use actions to compute (into a global variable) the length of the constant, and then count the length down on the right hand side. It would be fairly easy to process Hollerith constants in a hand written scanner, using the technique just described.

2. (45%) Consider the following grammar:

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

(a) (5%) Give the LL(1) parse table for this grammar.
Answer:

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

(b) (10%) Is the grammar LL(1) or not? Explain your answer. If not, transform the grammar into one that *is* LL(1), and give the LL(1) parse table for the transformed grammar.
Answer:

There is a conflict in entry $[A, y]$ so the grammar is not LL(1). Left-factoring produces the following grammar:

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow xA \mid yB \\ B &\rightarrow A \mid \varepsilon \end{aligned}$$

	x	y	\$
S	$S \rightarrow A$	$S \rightarrow A$	
A	$A \rightarrow xA$	$A \rightarrow yB$	
B	$B \rightarrow A$	$B \rightarrow A$	$B \rightarrow \varepsilon$

(c) (10%) Construct the LR(0) item sets for the original grammar. Is the grammar LR(0)? Why or why not?
Answer:

$$\begin{aligned} I_0 &= \{[S \rightarrow \bullet A], [A \rightarrow \bullet xA], [A \rightarrow \bullet yA], [A \rightarrow \bullet y]\} \\ I_1 &= \text{GOTO}(I_0, A) = \{[S \rightarrow A\bullet]\} \\ I_2 &= \text{GOTO}(I_0, y) = \{[A \rightarrow y \bullet A], [A \rightarrow y\bullet], [A \rightarrow \bullet xA], [A \rightarrow \bullet yA], [A \rightarrow \bullet y]\} \\ I_3 &= \text{GOTO}(I_0, x) = \{[A \rightarrow x \bullet A], [A \rightarrow \bullet xA], [A \rightarrow \bullet yA], [A \rightarrow \bullet y]\} \\ I_4 &= \text{GOTO}(I_2, A) = \{[A \rightarrow yA\bullet]\} \\ I_5 &= \text{GOTO}(I_3, A) = \{[A \rightarrow xA\bullet]\} \\ \text{GOTO}(I_2, y) &= \text{GOTO}(I_3, y) = I_2 \\ \text{GOTO}(I_2, x) &= \text{GOTO}(I_3, x) = I_3 \end{aligned}$$

The grammar is not LR(0). There is a SHIFT-REDUCE conflict in I_2.

(d) (10%) Construct the SLR(1) parse table for the original grammar. Is the grammar SLR(1)? Why or why not?
Answer:

	ACTION			GOTO	
	x	y	\$	S	A
0	$s3$	$s2$			1
1			a		
2	$s3$	$s2$	$r_{A\to y}$		4
3					5
4			$r_{A\to yA}$		
5			$r_{A\to xA}$		

Yes, the grammar is SLR(1). There are no conflicts in the parse table.

(e) (5%) Is the grammar LR(1)? Why or why not?
Answer:

Yes, the grammar is LR(1) since every SLR(1) grammar is LR(1).

(f) (5%) Exhibit an *unambiguous* grammar that is not LR(1).
Answer:

There are infinitely many unambiguous non-LR(1) grammars. Here is one of the simpler ones:

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

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 resolved deterministically.

3. (10%) Consider the following grammar:

$$\begin{aligned} S &\rightarrow X \\ X &\rightarrow YaYb \mid ZbZa \\ Y &\rightarrow \varepsilon \\ Z &\rightarrow \varepsilon \end{aligned}$$

(a) (5%) Is this grammar LL(1)? Why or why not?
Answer:

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.

(b) (5%) Is it SLR(1)? Why or why not?
Answer:

The grammar is not SLR(1), 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

$$[X \rightarrow \bullet YaYb], [X \rightarrow \bullet ZbZa], [Y \rightarrow \bullet], [Z \rightarrow \bullet]$$

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).

4. (15%) Consider the following grammar:

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

(a) (5%) Draw the parse tree for the input $y+++y++$.
Answer:

(b) (10%) Show the steps of an LR(1) parse of the input $y+++y++$, showing the input as it is consumed, the parse stack (left is bottom, right is top) at each step of the parse, and the action applied at each step.
Answer:

Stack	Input	Action
$\$$	$y+++y++\$$	s
$\$y$	$+++y++\$$	$r_{B\rightarrow y}$
$\$B$	$+++y++\$$	s
$\$B+$	$++y++\$$	s
$\$B++$	$+y++\$$	s
$\$B++$	$+y++\$$	$r_{A\rightarrow B++}$
$\$A$	$+y++\$$	s
$\$A+$	$y++\$$	s
$\$A+y$	$++\$$	$r_{B\rightarrow y}$
$\$A+B$	$++\$$	s
$\$A+B+$	$+\$$	s
$\$A+B++$	$\$$	$r_{A\rightarrow B++}$
$\$A+A$	$\$$	$r_{A\rightarrow A+A}$
$\$A$	$\$$	$r_{S\rightarrow A}$
$\$S$	$\$$	a

5. (20%) The following grammar describes a fragment of the syntax of an M3-like language using the same conventions as in the project. The productions for `AssignSt` are left unspecified. `Id` represents a variable name and `Number` represents an integer literal.

```
Program = Stmts.
Stmts = Stmt Stmts | Stmt.
Stmt  = AssignSt | ForSt.
ForSt = FOR Id ":=" Number TO Number DO Stmts END ";".
```

(a) (5%) Draw the parse tree for the following statements:

```
FOR i := 1 TO 100 DO
  AssignSt
  AssignSt
END;
AssignSt
```

(b) (10%) Write an attribute grammar that computes the number of executed statements for a program conforming to this grammar. For each attribute in your attribute grammar, say whether it is *synthesized* or *inherited*.
Answer:

The attributes are cnt and val. Both are synthesized attributes because their values are propagated up the tree.

`Program = PROCEDURE Stmts.`	`Program`.cnt = `Stmts`.cnt
$\texttt{Stmts}_0$ `= Stmt` $\texttt{Stmts}_1$`.`	$\texttt{Stmts}_0$.cnt = `Stmt`.cnt + $\texttt{Stmts}_1$.cnt
`Stmts = Stmt.`	`Stmts`.cnt = `Stmt`.cnt
`Stmt = ForSt.`	`Stmt`.cnt = `ForSt`.cnt
`Stmt = AssignSt.`	`Stmt`.cnt = 1

`ForSt =`
`FOR Id ":="`
$\texttt{Number}_0$ `TO` $\texttt{Number}_1$
`DO Stmts END ";".`
	`ForSt`.cnt = `Stmts`.cnt
	$\times \max(0, \texttt{Number}_2\text{.val} -$
	$\texttt{Number}_1\text{.val}+1)$

(c) (5%) Annotate the parse tree for the program fragment of question 5a with the computed attribute values. [You need not redraw the parse tree here. Simply add to your drawing of the parse tree in 5a.]