

Name: \_\_\_\_\_

USC ID: \_\_\_\_\_

## Midterm Exam CS 410, Spring 2002 [Bono]

February 28, 2002

There are 7 problems on the exam, with 80 points total available (that's about a point a minute). There are 8 pages to the exam, including this one; make sure you have all of them. If you need additional space to write any answers, you may use the backs of pages (just direct us where to look).

Put your name and SSN at the top of the exam. Please read over the whole test before beginning. Good luck!

	<b>value</b>	<b>score</b>
Problem 1	10 pts.	
Problem 2	7 pts.	
Problem 3	10 pts.	
Problem 4	5 pts.	
Problem 5	15 pts.	
Problem 6	23 pts.	
Problem 7	10 pts.	
<b>TOTAL</b>	80 pts.	

## Problem 1 [10 pts.]

**Part A [2].** The actions of a LL(1) parser correspond to a

\_\_\_\_\_ derivation.

**Part B [2].** `flex` is a tool for automatically building a

\_\_\_\_\_

from \_\_\_\_\_.

**Part C [2].** `bison` is a tool for automatically building a

\_\_\_\_\_

from \_\_\_\_\_.

**Part D [2].** When running a SLR(1) parser, suppose on top of the stack we have a handle for the production

$A \rightarrow \alpha$

for what inputs can we reduce by this production?

**Part E [2].** What is another name for a *reverse derivation step*?

## Problem 2 [7 pts.]

Consider the following regular expression over the alphabet  $\{a, b\}$  (Note:  $\epsilon$  denotes the empty string):

$$(b \mid \epsilon) (ab)^* (a \mid \epsilon)$$

**Part A [4].** Give a concise description (in English) of the *language* described by the regular expression. No credit will be given for writing down an English translation of the regular expression.

**Part B [3].** Use Thompson's construction to make an NFA from the regular expression (show it as a state diagram). You are not required to number the states, but do make sure it's clear what the start and end states are. Little or no credit will be given for creating an ad hoc NFA

**Problem 3 [10 pts.]**

Use subset construction to build a DFA equivalent to the NFA below. Show your work.

	<b>a</b>	<b>b</b>	$\epsilon$
<b>1</b>	{ }	{ 2 }	{ 2 }
<b>2</b>	{ 3 }	{ }	{ 4 }
<b>3</b>	{ }	{ 4 }	{ }
<b>4</b>	{ 5 }	{ }	{ 2, 5 }
<b>5</b>	{ }	{ }	{ }

**Problem 4 [5 pts.]**

Show that the following grammar is ambiguous. (Note:  $\epsilon$  denotes the empty string.)

$$\begin{aligned} A &\rightarrow x A y A \\ &| y A x A \\ &| \epsilon \end{aligned}$$

### Problem 5 [15 pts.]

Consider the following grammar over the alphabet  $\{w, x, y, z\}$ :

$$\begin{aligned} A &\rightarrow A x y \\ &\quad | B A y \\ &\quad | C \end{aligned}$$
$$\begin{aligned} B &\rightarrow B z x \\ &\quad | x \end{aligned}$$
$$\begin{aligned} C &\rightarrow w C \\ &\quad | y C \\ &\quad | z \end{aligned}$$

**Part A.** Do any grammar transformations necessary to make the grammar suitable for LL(1) parsing. Label the resulting grammar(s) with the transformation(s) you applied (or say “same” if no transformations were necessary).

**Part B.** Attempt to create the LL(1) parse table for the final grammar you gave in part A. Show the complete table: if this grammar is not LL(1), that is if there are conflicts, show all the values that could go in a table entry.

Show your work.

Circle and label your answers to each of the parts.

### Problem 6 [23 pts.]

Consider the following ambiguous expression grammar over the alphabet { **id**, **-**, **(**, **)** }:

$E' \rightarrow E$

$E \rightarrow E - E$

$E \rightarrow \mathbf{id}$

$E \rightarrow ( E )$

**Part A [15].** Show the DFA for recognizing viable prefixes for this grammar, including the sets of items associated with each state; show this as a state diagram. Number each of the states (such that 0 is the start state). Note: we have already augmented the grammar for you.

**Part B [5].** If we attempt to build an SLR parser with the grammar given above, we'll get a conflict. Show exactly where and how this error occurs (i.e., in terms of states from part A, inputs, and the relevant entry in the action table).

**Part C [3].** Explain how the parser should resolve the conflict such that we get the normal interpretation for '-' expressions (in, for example, C, and many other programming languages).

### Problem 7 [10 pts.]

Add semantic rules to the following grammar to compute the attribute **rm**, whose value is the rightmost terminal in the string we parsed. For example, if the string parsed were  $zxyxy$ ,  $S.rm$  would be  $y$ .

Note: subscripts in the grammar below are only to distinguish multiple instances of the same nonterminal.

---

$S \rightarrow A \quad \{ S.rm =$

$A \rightarrow A_1 \mathbf{x} \mathbf{y}$

|  $B A_1 \mathbf{y}$

|  $C$

$B \rightarrow B_1 \mathbf{z}$

|  $\mathbf{x}$

$C \rightarrow \mathbf{w} C_1$

|  $\mathbf{y} C_1$

|  $\mathbf{z}$