

Name: _____

USC loginid (no SSN): _____

CS 410 Midterm Exam Spring 2004 [Bono]

March 8, 2004

There are 5 problems (plus one extra credit problem) on the exam, with 50 points total available. There are 7 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 to look there).

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

	value	score
Problem 1	4 pts.	
Problem 2	10 pts.	
Problem 3	15 pts.	
Problem 4	13 pts.	
Problem 5	8 pts.	
<i>Extra Credit problem</i>	<i>5 pts.</i>	
TOTAL	50 pts.	

Problem 1 [4 pts.]

Short answer.

A [1]. Thompson's construction is a method for automatically building a

_____ from a _____.

B [1]. *bison* creates a (circle the one that best matches)

- a. grammar
- b. bottom-up parser
- c. abstract syntax tree
- d. top-down parser
- e. regular expression

C [2]. Suppose we have the a state in the DFA for an SLR parser corresponding to the following set of items (assume lower case letters are terminals, and the dot is to show the item):

[A --> C . y B]
[B --> z D .]

The resulting action table would have a shift-reduce conflict in this state if

_____ (a symbol from the grammar)

is an element of _____ .

Problem 2 [10 pts.]

Use subset construction to build a DFA equivalent to the NFA below. Show your work. Note: ϵ is the epsilon symbol.

	a	b	ϵ
1	{ 2 }	{ }	{ 3 }
2	{ }	{ 3 }	{ }
3	{ }	{ }	{ 1, 5 }
4	{ }	{ }	{ }
<u>5</u>	{ }	{ 5 }	{ }

Problem 4 [13 pts.]

Consider the language of zero or more *comma-separated expressions*. Here are four sample sentential forms in the language:

ϵ *expr* *expr* , *expr* *expr* , *expr* , *expr*

where *expr* could expand to some expression and ϵ is the empty string.

Part A [5]. Suppose someone proposes the following grammar for the language:

```
list  $\rightarrow$   $\epsilon$ 
      |  expr
      |  list , expr
```

Explain why the grammar given does not denote the language described.

Part B [5]. Write a correct grammar for the language described.

Part C [3]. Write a regular expression for the language described. For the purposes of this part of the problem consider **expr** as part of the input alphabet.

Problem 5 [8 pts.]

Write semantic rules for the following expression grammar such that they translate the parsed infix expression to the equivalent lisp expression. By translate, we mean that the semantic attribute will be of type `string`. Lisp uses a fully parenthesized prefix notation for expressions. For example:

infix form	lisp form
	<i>(i.e, what will be in the string <code>E.lisp</code> after parsing the infix expr)</i>
<code>a + b</code>	<code>(+ a b)</code>
<code>a * b + c</code>	<code>(+ (* a b) c)</code>
<code>a * b * c + d</code>	<code>(+ (* (* a b) c) d)</code>

Reminder of C++ standard library `string` operations

```
string s = "big"; // assign a C-string to a string
string t = " dog";
string u = t; // assign one string to another
u = s + t; // concat strings using overloaded +
// u has the value "big dog"
u = s + " bad" + t; // concat string with a C-string
// u has the value "big bad dog"
u[0] = 'd'; // access / change individual chars
// u has the value "dig bad dog"
```

For the attributes you may use either bison syntax or named semantic attributes. If you do the latter, call your attribute `lisp`. Assume that `id.lisp` is a `string` containing the identifier read. Note: The subscripts below are only to distinguish multiple instances of a non-terminal in a production.

`E --> E1 + T`

| T

`T --> T1 * F`

| F

`F --> id`

| (E)

Extra Credit Problem [5 pts.]

NOTE: Please don't spend time on this problem unless you have completed the rest of the exam.

Lisp addition expressions are allowed to take more than two operands (we'll call that *multi-plus* here). Here is an example of it:

infix form	lisp multi-plus form
a + b + c + d	(+ a b c d)

Using the language described by the grammar below, *write a new grammar* for this language along with semantic rules to translate an infix plus expression to the multi-plus Lisp form. As in the last problem, the translation will be a `string` semantic attribute.

```
plusExpr --> id + plusExpr
          | id + id
```