Name: _____

USC loginid (e.g., `ttrojan`): _____

CS 410 Midterm Exam
Fall 2004 [Bono]

October 15, 2004

There are 6 problems on the exam, with 50 points total available. There are 6 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	3 pts.	
Problem 2	5 pts.	
Problem 3	9 pts.	
Problem 4	9 pts.	
Problem 5	17 pts.	
Problem 6	7 pts.	
TOTAL	50 pts.	

Problem 1 [3 pts.]

Give an example of a language that is **not** *regular*. You may describe the given language in words, or using some more formal notation. Your answer may not be a human language (such as Korean or Esperanto).

Problem 2 [5 pts.]

Write a grammar for the following language:

```
(a|b)*c(a|b)*
```

Problem 3 [9 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	{ 1 }	{ 2 }	{ }
2	{ 4 }	{ 3 }	{ 4 }
3	{ 3 }	{ 3,4 }	{ }
4	{ }	{ }	{ 3 }

Problem 4 [9 pts.]

Each line shown in **bold** in the complete Espresso program below contains an error. Next to each bold line:

- say which part of the compiler would detect and report the error. You can use the following abbreviations:

  L = lexical analyzer

  P = parser

  S = semantic analyzer

  C = code generator

- say what the error is

You may assume that each error is independent, and that the compiler recovers and continues processing the rest of the program after an error occurs.

```
class Main {
   public static void main(String[] args)
   {
      Foo x;



      int y;
      int z;
      y = 0;
      z = 0;


      if (y === 3) y = 10;



      y = z.foo();



      y = 345xyz;



      y = xyz345;



      y = y @ z;
  }
}
```

Problem 5 [17 pts. total]

Part A [10]. Build the DFA for recognizing viable prefixes for the grammar below. Show the set of items for each state. Note: the grammar has already been augmented for you. You may use the space below, or the back of the previous page (just direct us to look there).

Part B [5]. Using the grammar, and your work from part **A**, complete ***only the reduce and accept entries*** for the action table for an SLR parser for this grammar. (Do not put in the shift or goto table entries.) You must show your work. We made the grid for you to save you some time. You won't necessarily need all the rows displayed.

0) `S'` → `S`

1) `S` → `( A )`

2) `S` → `( S )`

3) `A` → **`a`**

4) `A` → **`b`**

ACTION TABLE

state	(	)	a	b	$

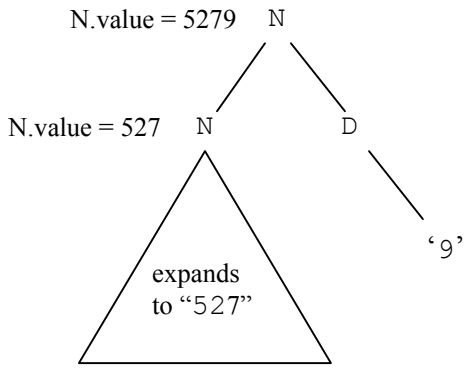
Problem 6 [7 pts.]

Consider the grammar for an integer constant at the bottom of the page (nonterminal `N` stands for number, and `D` for digit).

Write semantic rules for the grammar to compute the numeric value of the integer constant. We'll call this value `N.value`. Your rules may only use semantic attributes for data; no global variables.

You do not have to write actions for the elided (. . .) productions.

HINT to help you think recursively for this problem: Consider the sentence "`5279`". The partially expanded parse tree below shows the top-level derivation steps and the semantic values of two different instances of `N` in the derivation (Note: not all semantic values in the tree are shown).




```
N -->  N1 D

    |  D


D --> '0'

    | '1'

      .
      .
      .

    | '9'
```