

CSCE 531
Spring 2009
Final Exam

Do all problems. Write your solutions on the paper provided. This test is open book, open notes, but no electronic devices. For your own sake, please read *all* problems before trying *any* of them; the point value of a problem may not be commensurate with its difficulty.

You need not show your work unless explicitly asked to do so, but if your answer is incorrect, you are more likely to get partial credit if you show your work.

There are 100 points total in the exam. Any points earned beyond those limits counts as extra credit.

1. (10 points) Using the method described in the textbook or in class, convert the regular expression

$(ab|c)^*$

into an equivalent NFA (which can have ε -moves).

2. (15 points total) A *list* is a finite sequence of things surrounded by parentheses. For example, $(1\ 4\ 9\ 16)$ is a list of the first four perfect squares. (Items in the list are separated by whitespace if necessary.) Anything that is not a list is called an *atom*, e.g., numbers, symbols, strings, etc. The items of a list could be other lists, e.g., $((1\ 1)\ (1\ 2)\ (2\ 1)\ (2\ 2))$. The empty list $()$ with no items is also a list.

A *generalized list* is either an atom or a list whose items (if any) are themselves generalized lists. This is recursive definition, of course. So a generalized list could be a list of lists of lists

- (a) (5 points) Assuming three terminal symbols: $(,)$, and **atom**, give a grammar for a list of atoms suitable for bottom-up parsing. Express your grammar in abstract form, with start symbol A .
- (b) (10 points) Assuming three terminal symbols: $(,)$, and **atom**, give a grammar for generalized lists suitable for bottom-up parsing. Express your grammar in abstract form, with start symbol B .

In both cases, you should assume that any whitespace is consumed by the lexical analyzer to separate items and is not itself a token. [Hint: Both answers are rather short.]

3. (20 points) Consider the following grammar with start symbol S' for all strings of matching parentheses:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow S(S) \\ S &\rightarrow \varepsilon \end{aligned}$$

Using the method described in class or the text, construct the set of states for a canonical LR(1) parser for this grammar, defining the transition function at the same time. Note that in class I denoted the transition function as *trans*. The textbook denotes the same function as *goto*.

To ensure a unique correct answer, you must stick to the following rules of order, which mirror the order I used for my example in class:

- Give each state as a list of LR(1) items, omitting the brackets.
 - Give the start state first, and denote it by s_0 . Denote other states s_1, s_2, \dots in the order they are constructed.
 - List the kernel items first in each state. List additional nonkernel items in the order that they enter the closure.
 - For each $i \geq 0$, define all transitions out of s_i before defining those out of s_{i+1} .
 - When finding the transitions out of a state, or computing a closure, consider each item of the state in the order you listed it.
 - Do not list the empty set as a state.
4. (25 points total) Consider the following fragment of bison code for expressions with constants TRUE and FALSE, and boolean operators & (AND), | (OR), and ! (NOT):

```
bool_expr:
    boolean_expr '|' conjunction
    | conjunction
    ;
```

```
conjunction:
    conjunction '&' literal
    | literal
```

```
literal:
    | '!' literal
    | primary
    ;
```

```
primary:
    TRUE
```

```

| FALSE
| '(' bool_expr ')',
;

```

(a) (10 points) Give a parse tree with root `bool_expr` for

```
FALSE | FALSE | !(TRUE & !TRUE)
```

(b) (15 points) Assuming each nonterminal has an integer attribute, with 0 representing FALSE and any nonzero value representing TRUE, add semantic actions to evaluate the truth value of a boolean expression. This evaluation takes place while the expression is parsed. Do not declare or use any variables besides \$-variables (semantic stack items).

5. (10 points) Consider the following Pascal declaration:

```

var
  a : array[3..7] of array[4..6] of Integer;

```

Assuming that a Pascal Integer is four bytes and that the base address of `a` is 1500, find the base address of the integer variable `a[6][5]`.

6. (20 points total) Consider the following three-address code (line numbers added):

```

1      L1:      i := a
2      L2:      if i <= 10 then goto L4
3          i := i + 1
4          goto L2
5      L3:      if j <= i then goto L5
6      L4:      j := j + 1
7          goto L3
8      L5:      a := a - 1
9          if a > 0 then goto L1
10

```

Assume that control enters at line 1 and that there are no other entry points.

- (5 points) Describe the basic blocks B_1, B_2, \dots by giving an inclusive range of line numbers for each block.
- (10 points) Draw the flow diagram as a directed graph, labeling the nodes B_1, B_2, \dots . Give dangling arrows both for the entry point and for the exit point of the code as a whole.
- (5 points) Using the strict definition of a loop as defined in class and in the text, list sets of vertices that constitute loops. Label any inner loop(s) as such.