Victoria University of Wellington

DEGREE EXAMINATIONS — 1998

END OF YEAR

COMP 202

Formal Methods of Computer Science

Time Allowed: 3 Hours

Instructions: Candidates should attempt **all** questions.

This exam will be marked out of 100.

Question 1. [15 marks]

Consider the regular expression $(aa)^*b \mid a(aa)^*c$.

(a) Construct a transition diagram for an NFA to recognise the language defined by this regular expression, using the "top-down construction". [5 marks]

You do not need to show all of the steps in the construction, but you **should** show all of the states and transitions resulting from using the top-down construction.

(b) Construct a transition diagram for a DFA, equivalent to your NFA in part **(a)**, using the "subset construction".

Show the relationship between states in your DFA and those in the NFA. [5 marks]

(c) Write a Regular Grammar, equivalent to the original regular expression, based on your DFA in part **(b)**. [5 marks]

continued...

Question 2. [40 marks]

Suppose you wish to write a program to solve sets of algebraic equations. The first thing you need is a parser that allows you to read a list of equations and build a representation of them which your solver can operate on.

Suppose we define the syntax of equation lists as follows:

$$\begin{array}{lcl} \textit{eq-list} & \rightarrow & \textit{eqn} \mid \textit{eq-list} \text{ ";" } \textit{eqn} \\ \textit{eqn} & \rightarrow & \textit{exp} \text{ "=" } \textit{exp} \\ \textit{exp} & \rightarrow & \textit{id} \mid \textit{id} \text{ "(" } \textit{exp-list} \text{ ")"} \\ \textit{exp-list} & \rightarrow & \textit{exp} \mid \textit{exp-list} \text{ "," } \textit{exp} \end{array}$$

where *id* is an identifier.

(a) Explain what it means for a grammar to be in LL(1) form, and why this property is important in the construction of recursive descent parsers. [5 marks]

(b) Identify any places where the above grammar fails to meet the LL(1) conditions. [3 marks]

(c) Write a (plain) CFG in LL(1) form, equivalent to the grammar above.

Show that your grammar is in LL(1) form, by constructing the necessary $first$ and $follows$ sets.

Explain why constructing an LL(1) grammar in this way often has undesirable consequences. [7 marks]

(d) Construct an LL(1) parser table, based on your grammar in part **(c)**. [5 marks]

(e) Extended Context Free Grammars (ECFGs) provide an attractive basis for constructing recursive descent parsers, because repetition can be handled by loops, both in the grammar and in the parser.

Write an ECFG, in LL(1) form, equivalent to the grammar given above.

You should make best use of the features of ECFGs to obtain a compact and intelligible grammar, reflecting the structure of the original grammar as closely as possible. [5 marks]

(f) Write the procedures required for a recursive descent parser to recognise lists of equations, based on your grammar from part **(e)**.

You should assume the availability of a scanner, recognising the terminal symbols used in this grammar (i.e. ";", "=", "(", ")", "," and *id*). [10 marks]

(g) Explain how you would extend your parser in part **(f)** to build a representation of the list of equations recognised.

You may assume any reasonable representation for lists of equations, provided you explain the operations used in its construction. [5 marks]

Question 3. [25 marks]

(a) Explain briefly the difference between *strong equivalence* and *weak equivalence* of programs. [4 marks]

(b) Consider the following three while programs, where B_1 and B_2 are any Boolean expressions and S_1, S_2, S_3 are any statements:

```
P_1:  if $B_1$ then $S_1$ else $S_2$ fi;
      if $B_2$ then $S_3$ else $S_4$ fi
```

```
P_2:  if $B_1$ then
          $S_1$; if $B_2$ then $S_3$ else $S_4$ fi
      else
          $S_2$; if $B_2$ then $S_3$ else $S_4$ fi
      fi
```

```
P_3:  if $B_1$ then
          if $B_2$ then $S_1$; $S_3$ else $S_1$; $S_4$ fi
      else
          if $B_2$ then $S_2$; $S_3$ else $S_2$; $S_4$ fi
      fi
```

(i) Show that P_1 and P_2 are *strongly* equivalent. [4 marks]

(ii) Under what circumstances will P_1 and P_3 be *weakly* equivalent? [2 marks]

(c) Show that, for any Boolean expression B and any statements S_1 and S_2, there is a while program which is *strongly* equivalent to the following flowchart program: [5 marks]

```
1:  $S_1$;
    if $B$ then 2 else 3;
2:  $S_2$;
    goto 1;
3:  skip
```

(d) Show that, for any while program, there is a *weakly* equivalent while program containing only one **while** statement. [5 marks]

(You may assume results about program transformations that were proved in the course notes.)

(e) Demonstrate the effect of the transformation you described in part **(d)** on the following while program: [5 marks]

```
$k := 1$;
while $n \neq 0$ do
    while $A[k] \neq x$ do
        $k := k + 1$
    od;
    $n := n - 1$; $k := k + 1$
od
```

continued...

Question 4. [20 marks]

The following is an algorithm to count the number of index positions at which two strings s and t have identical elements, assuming that $|s| = m$ and $|t| = n$.

```
$k := 0$; $c := 0$;
**while** $k \neq m \ \wedge \ k \neq n$ **do**
    **if** $s[k+1] = t[k+1]$ **then**
        $c := c + 1$
    **fi**;
    $k := k + 1$
**od**
```

If we write $count(i,\ S,\ P)$ for the number of elements, i, in set S, satisfying property P (i.e. $count(i, S, P) = |\{\ i \mid i \in S \ \wedge \ P\ \}|$), and $u\,..\,v$ for the set $\{j \mid u \leq j \leq v\}$, we can express the postcondition for this program as:

$$c = count(\ i,\ 1\,..\,min(m, n),\ s[i] = t[i]\ )$$

(a) Explain what is meant by a "loop invariant", and how a loop invariant can be used in proving that a loop satisfies a given specification. [5 marks]

(b) Give a loop invariant that could be used to verify the loop in the above program. [5 marks]

(c) Use your loop invariant from part **(b)** to prove that the program correctly computes the number of index positions at which s and t have identical elements.

You should give the verification conditions that must hold in order for the program to be correct, and clearly identify any mathematical properties of strings and/or of counting that your proof relies upon. [10 marks]

****