VICTORIA UNIVERSITY OF WELLINGTON

Te Whare Wananga o te Upoko o te Ika a Maui

EXAMINATIONS — 2000

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.
Foreign language dictionaries are permitted.

Question 1. [30 marks]

(a) Consider the two regular expressions:

$$R_1 = 0^*|1^* \text{ and } R_2 = 01^*|10^*|1^*0|(0^*1)^*$$

(i) Find a string that is in $L(R_1)$ but not in $L(R_2)$.

(ii) Find a string that is in $L(R_2)$ but not in $L(R_1)$.

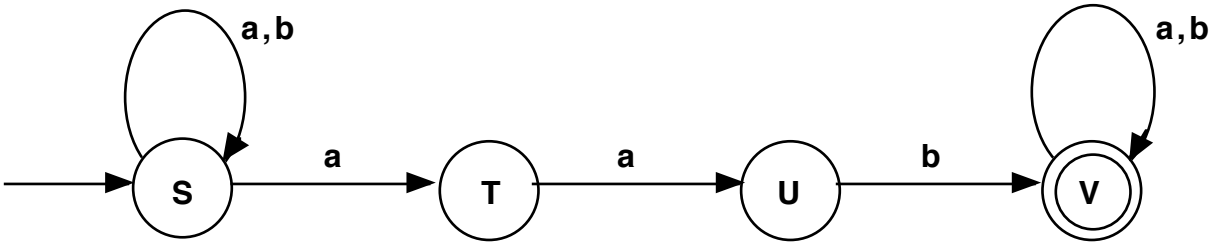
(iii) Find a string that is in both $L(R_1)$ and $L(R_2)$.

(iv) Find a string in $\{0,1\}^*$ that is neither in $L(R_1)$ nor in $L(R_2)$.

[4 marks]

(b) The following NFA recognises the language defined by the regular expression $(a|b)^*aab(a|b)^*$.

That is, the set of strings in $\{a,b\}^*$ that contain the substring aab.



Use this NFA and the subset construction to design a (complete) DFA that recognises the set of strings in $\{a, b\}^*$ that **do not** contain the substring *aab*. Show clearly which subset of states from the NFA corresponds to each state of your DFA. Show clearly which states of your DFA are final/accepting states. [5 marks]

(c) Number the states in your DFA, and apply the marking algorithm given in lectures to find the minimal DFA that recognises the same language as your DFA. Show your working in applying the marking algorithm, and draw the resulting minimal DFA. [4 marks]

(d) In the *C* programming language, all the following expressions represent valid numerical "literals":

3	13.	.328	41.16	+45.80
+0	−014	−14.44	41*e*124	+1.4*e*6
−2.*e* + 7	01*E* − 06	0.2*E* − 20	−.4*E* − 7	00*e*0

Either '*e*' or '*E*' refers to an exponent, and if it appears, it must be followed by an integer. Based on these examples, write a regular expression representing the language of numeric literals. You can write: *d* for digit, *a* for "plus", *m* for "minus", and *p* for "point". Assume that there is no maximum limit on the number of consecutive digits in any part of the expression. [6 marks]

(e)(i) Let L be the set of even-length strings over $\{0, 1\}$ whose first and second halves are identical. That is, $L = \{ww \mid w \in \{0, 1\}^*\}$. Show that for any two distinct strings x and y in $\{0, 1\}^*$, x and y are distinguishable with respect to L.

(ii) Let L be the language $\{0^n 1^n \mid n \geq 0\}$. Show that there are distinct strings x and y in $\{0, 1\}^*$ that are indistinguishable with respect to L.

[6 marks]

(f) Consider the set of strings over the single symbol alphabet $\{a\}$ whose length is a power of 2. That is, $L = \{x \in \{a\}^* \mid |x| = 2^n, n \geq 0\}$.

Either:

Use the Myhill-Nerode theorem to prove that L is not regular.

You can use, without proof, the following facts:

$2^k + 2^k = 2^{k+1}$
$2^m + 2^n \neq 2^k$ for any k, if $m \neq n$

Or:

Prove that the set of lengths of strings in L, $\{m \mid a^m \in L\}$, is not ultimately periodic, and hence L is not regular.

A subset of the natural numbers, S, is ultimately periodic if there exist numbers $n \geq 0$ and $p > 0$ such that for all $m \geq n$, $m \in S$ if and only if $m + p \in S$.

[5 marks]

Question 2. [40 marks]

(a) Explain what it means for a Context Free Grammar to be *ambiguous*. [4 marks]

(b) Explain what it means for a Context Free Grammar to be in *LL(1) form*, and why this property is important in the construction of recursive descent parsers. [4 marks]

The Hypertext Markup Language (HTML) is used to describe the structure of documents to be displayed by a Web browser.

The following constructs are a (rather small) subset of those used in the language.

Every HTML document should begin by declaring itself as such, using the `<html>` tag at the very top of the document, and corresponding ending tag `</html>` as the very last text in the document.

The first part of the document should be a section for heading information, surrounded by `<head>` and `</head>` tags. Several items of information may go in the header, but it should always contain the title, surrounded by `<title>` and `</title>` tags.

The second part of the document is the body, surrounded by `<body>` and `</body>` tags.

The body may contain (any number of) headings, paragraphs, and lists (among other items), repeated in any order. We will assume only these three constructs.

Headings are surrounded by heading tags `<hX>` and `</hX>`, where `X` is the heading level. We will assume only one level for headings, and will only allow plain text between the heading tags.

For our purposes, paragraphs are surrounded by `<p>` and `</p>` tags and contain only plain text.

We will assume only one type of list, an ordered list, which begins with the tag `<ol>` and ends with the tag `</ol>`.

A list may contain any number of list items. List items begin with the `<li>` tag. For our purposes, a list item may be either a block of plain text or another list. Thus, lists may be nested by including a list as a list item.

Here is a sample document, illustrating the constructs described:

```
<html>
<head>
<title>This is my title</title>
</head>
<body>

<h1>This is my main heading</h1>

<p>This is a sample paragraph. The majority of documents contain
this type of construct. </p>
```

```
<h1>This is another heading</h1>

<p> The quick brown fox jumped over the slow lazy dogs.
The quick brown fox jumped over the slow lazy dogs.</p>

<p>Here's an ordered list:</p>
<ol>
  <li> an item.
  <li> another item.
  <li> next item is a nested list
  <li>
    <ol>
      <li> a nested item
      <li> another nested item
    </ol>
  <li> the last item
</ol>

</body>
</html>
```

(c) Write a (plain) Context Free Grammar, in LL(1) form, to recognise the **body** of an HTML document, as described above. You should assume the availability of a scanner, recognising HTML commands as terminal symbols in the grammar, and recognising blocks of text not containing any HTML commands as corresponding to the special terminal symbol *textblock*. [6 marks]

(d) Show that your grammar is in LL(1) form by constructing the necessary *first* and *follow* sets. [6 marks]

(e) Explain why constructing an LL(1) grammar in this way sometimes has undesirable consequences. [4 marks]

(f) Using an Extended Context Free Grammar (ECFG) as a basis for constructing a recursive descent parser means that repetition and alternation can be handled more elegantly.

Write an ECFG, in LL(1) form, equivalent to your grammar from part **(c)**.

You should make the best use of the features of ECFG's to obtain a compact and intelligible grammar, reflecting the structure of the HTML constructs. [6 marks]

(g) Write the procedures required for a recursive descent parser to recognise the body part of HTML documents, based on your ECFG. [10 marks]

Question 3. [15 marks]

(a) Consider the following while program (where / means integer division):

$u := x$; $v := y$; $z := 1$;
while $v \neq 0$ **do**
  if $(v/2) \times 2 = v$ **then**
    $u := u \times u$; $v := v/2$
  else
    $z := z \times u$; $v := v - 1$
  fi
od

(i) Write a regular expression denoting the set of all potential execution paths through this program. [2 marks]

(ii) Translate this program into a strongly equivalent flowchart program, using the construction given in the course notes. [6 marks]

(b) We can translate any while program, P, into a while program, P', containing no **if** statements, by applying the following transformation to each **if** statement:

if B **then** S_1 **else** S_2 **fi** $\rightarrow$ $b := 1$;
while $B \wedge b = 1$ **do** S_1'; $b := 0$ **od**;
while $b = 1$ **do** S_2'; $b := 0$ **od**

where a new variable b, not appearing anywhere else in the program, is used for each **if** statement transformed, and S_1' and S_2' are the results of applying this transformation to S_1 and S_2, respectively.

(i) Show that P' is not strongly equivalent to P. [2 marks]

(ii) Show that P' is weakly equivalent to P. [5 marks]

Question 4. [15 marks]

The following is an algorithm to decide whether the elements in an array $A[\,1..n\,]$ are in ascending order.

```
input A, n;
{ (1 ≤ n) ∧ (n = |A|) }
j := 1;
b := true;
while j ≠ n do
    if b then
        if A[j + 1] < A[j] then b := false fi
    fi;
    j := j + 1
od;
output b
```

$\{\ b \iff asc\{A[1]..A[n]\}\ \}$

.

$\{\ (b \wedge \forall i \in \{1, ..., n-1\}, A[i] \leq A[i+1]) \vee (\neg b \wedge \exists i \in \{1, ..., n-1\}, A[i] > A[i+1])\ \}$

.

$\{\ (b \iff asc\{A[1]..A[j]) \wedge (1 \leq j \leq n)\ \}$

(a) A precondition is given which is assumed to hold just after the **input** statement. Give a sensible postcondition to complete the specification. [4 marks]

(b) Give a loop invariant that could be used to verify the loop. [4 marks]

(c) Show that the following verification conditions hold:

(i) The loop invariant holds on entry to the loop.

(ii) The loop invariant is preserved by the loop body.

(iii) The postcondition that you specified in part **(a)** holds when the loop exits with the loop invariant true.

[7 marks]

****