OLLSCOIL NA hÉIREANN
THE NATIONAL UNIVERSITY OF IRELAND, CORK

COLÁISTE NA hOLLSCOILE, CORCAIGH
UNIVERSITY COLLEGE, CORK

Summer Examinations 2013

CS4150: Principles of Compilation

Professor I. Gent
Professor B O'Sullivan
Dr K. T. Herley

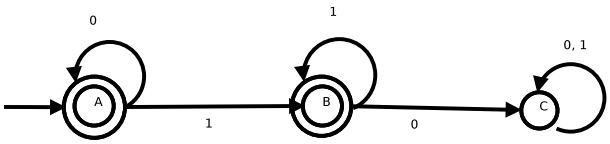
Answer All Questions
Total marks 100%

1.5 Hours

Question 1 *[30%]*

(i) Give a deterministic finite automaton for the set of binary strings such that every block of three consecutive bits contains at least one 1. (Hint: Use the automaton states to represent the two preceding bits at each stage.) *(8%)*

(ii) Describe in English the set of strings accepted by the automaton depicted below. Give a regular expression for the set of strings.



(6%)

(iii) Give a regular expressions for the set of all binary strings that do not contain a pair of adjacent 0's. *(7%)*

(iv) Construct a finite automaton equivalent to the following regular expression $10|(0|11)0*1$. *(9%)*

Question 2 *[20%]*

(i) Formulate a context free grammar for the following simple subset of HTML. Documents contain no text or content and involve the following tags only: html, head, title, body, h1, p, ol, li. Documents also contain no attributes, CSS or scripts– just tags drawn from the list above.

The grammar should respect as faithfully as possible the standard HTML constraints on the ordering, sequencing and nesting of such tags.

The body of the document may contain any number of paragraphs. Each paragraph may optionally contain one or more lists. List items may optionally contain other lists. *(15%)*

(ii) Give a complete parse tree for the following document. *(5%)*

```
<html>
   <head><title></title></head>
<body>
<p></p>
<p>
   <ol> <li></li> <li></li> <li></li> </ol>
</p>
</body>
</html>
```

Question 3 *[35 %]*

(i) List the FIRST sets of the non-terminals of the grammar shown below that is based on the venerable programming language Pascal.

⟨program⟩ → **PROGRAM ID ;** ⟨compound-statement⟩ ⊙
⟨compound-statement⟩ → **BEGIN** ⟨optional-statements⟩ **END**
⟨optional-statements⟩ → ⟨statement-list⟩
| ϵ
⟨statement-list⟩ → ⟨statement⟩ **;** ⟨statement-list⟩
| ϵ
⟨statement⟩ → ⟨variable⟩ **:=** ⟨expression⟩
| ⟨compound-statement⟩
| **IF** ⟨expression⟩ **THEN** ⟨statement⟩ **ELSE** ⟨statement⟩
| **WHILE** ⟨expression⟩ **DO** ⟨statement⟩
⟨variable⟩ → **ID**
⟨expression⟩ → **EXPR**

The non-terminals are enclosed within <>, while the terminals (reserved words PROGRAM, BEGIN, END, IF, THEN, ELSE, WHILE, DO; place-holders ID and EXPR and symbols ';', '⊙' and ':=') are shown in boldface. (*10%*)

(ii) List the FOLLOW sets of the non-terminals of this grammar. (*10%*)

(iii) Sketch the LL(1) table for this grammar. You need not compete the entire table but should indicate the row and column headings and the rows that correspond to the two non-terminals statement-list and statement. (*10%*)

(iv) Write a brief note indicating the connection between the stack-based LL(1) parsing technique mentioned above and the classical recursive descent parsing approach. (*5%*)

Question 4 *[15 %]*

(i) Write a brief note sketching how a parse/syntax tree might be used to facilitate a syntax-directed translation of a Pascal program into three-address code (TAC). You are not required to draw a tree merely to describe in words how a tree might drive the translation process. (A summary of TAC are appended to this paper.) (*3%*)

(ii) Show a complete translation of the following Pascal fragment into TAC, indicating clearly the connection between the various source constructs and the corresponding TAC. You may assume that the Pascal if, while and assignment statements share the same semantics as their Java equivalents and that Pascal's BEGIN-END play the same role as Java's {} in grouping statements together. (*12%*)

```
IF 0 < x THEN
   BEGIN
   fact := 1;
   WHILE x > 1 DO
      BEGIN
      fact := fact * x;
      x := x - 1
      END
   END
ELSE
   BEGIN
   END
```

Three-Address Code

x := y op z	assignment
x := op y	unary assignment
x := y	copy
goto L	unconditional jump
if x relop y goto L	conditional jump
param x	procedure call
call p n	procedure call
return y	procedure call
x := y[i]	indexed assignment
x[i] := y	indexed assignment