IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2003

LANGUAGE PROCESSORS

Tuesday, 10 June 2:00 pm

Time allowed: 2:00 hours

There are FOUR questions on this paper.

Q1 is compulsory.
Answer Q1 and any two of questions 2-4.

Q1 carries 40% of the marks. Questions 2 to 4 carry equal marks.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible First Marker(s) : Y.K. Demiris

Second Marker(s) : G.A. Constantinides

QUESTION 1:

(a) Describe five criteria that can be used to judge the quality of a language processor. [2]

(b) Briefly describe the types of grammar as defined in Chomsky's hierarchy of grammars, and provide the restrictions that each type imposes on its grammar production rules. [4]

(c) Provide the transition diagram for a push-down automaton that can be used to recognize the language $\{x^n y^n, n \geq 1\}$. [5]

(d) Give an example of a language that *cannot* be recognized by a push down automaton. [3]

(e) Describe how you can remove left-recursion from the production rules of a grammar. [3]

(f) Specify three techniques for code optimisation, and provide a code example for each. [3]

QUESTION 2:

For a programming language we have the following definitions:

- Identifiers must start with a letter, followed by zero or more letters or digits. Examples include *Temp1*, *index*, *a123*, among others.
- Numbers can be written either in decimal or scientific notation; the format consists of the following parts:
  - [mandatory] one or more digits
  - [optional] a decimal point, followed by one or more digits, optionally followed by "E", an optional plus or minus sign, and one or more digits.

  Examples include *26*, *1.234*, *5.2*, *34.4E2*, *6.3E-11*, among others.

(a) Write the regular expressions for valid identifiers and numbers for this language; define all special characters you used. [6]

(b) Provide a finite state automaton that, given an input string, will recognize it as either a valid identifier, or as a valid number. Clearly mark all final (accepting) states for the FSA. [14]

QUESTION 3:

Construct the deterministic finite automaton for the regular expression *(a|b)*c* by:

(a) constructing a non-deterministic finite automaton (NFA) using Thompson's algorithm. [10]

(b) constructing the equivalent DFA using the subset construction algorithm. Explain the intermediate steps you have taken. [10]

QUESTION 4:

(a) Describe the data structures involved, and the steps performed by the LR parsing algorithm. [10]

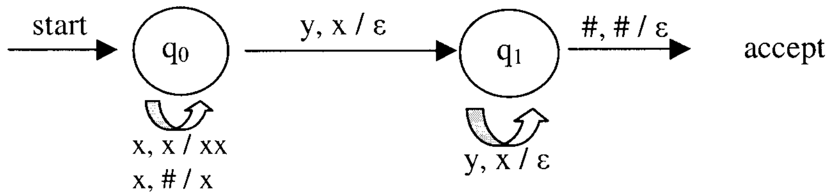
(b) Describe the algorithm for register allocation via graph colouring, including a heuristic algorithm for determining whether a graph *G* is colourable using a number of colours, *K*. [10]

E2.15: Language Processors
Model answers to exam questions 2003

Question 1

(a) [bookwork] Five criteria: correctness of generated code, conformity to the language specification, quality of generated code (size and speed), speed of language processor itself, and user-friendliness (for example the quality of its error reporting).
(b) [bookwork] Type 0, 1, 2, and 3; 0 (unrestricted grammars), 1 (for all productions α->β, we must have |α| ≤ |β|), 2 or context free grammars (only a single non-terminal may appear on the left-side of a production), and 3 or regular grammars (productions should all be left-linear or right linear)
(c) [new computed example]

(labels on the arrows: input symbol, stack symbol popped / symbols pushed)



(d) [bookwork] $\{x^n y^n z^n, n>=1\}$
(e) [bookwork] By changing all the rules of the form
A -> Aα | β
to
A-> β R
R -> α R | ε
(f) [bookwork] 1. Removal of unnecessary jumps:

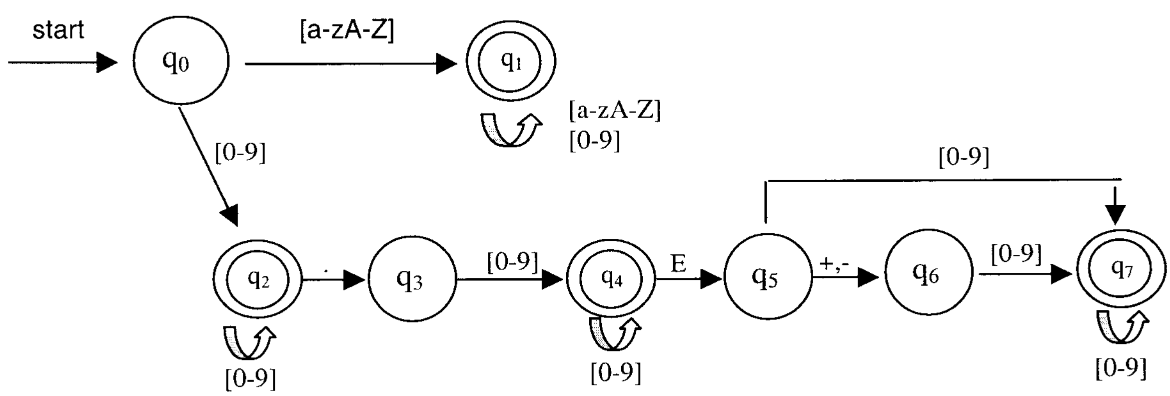
```
goto L1                       goto L2
...              ->           ...
L1:    goto L2                L1:    goto L2
```

2. Constant folding:
x = constant op constant
can be replaced by calculating the result of the operation and replacing the operation with the result
3. Moving loop invariant computations outside the loop

```
for I = 1 to 10              A = 2*pi;
  Begin                      for I=1 to 10
    A=2*pi         =>            Begin
    Z[I] = A;                         Z[I]=A;
  End                            End
```

Question 2
[new computed example]

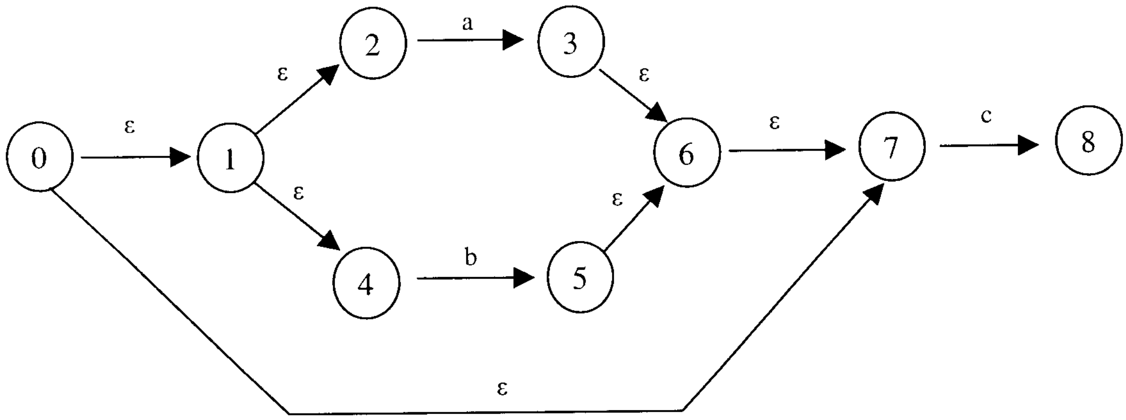
(a) identifier: [a-zA-Z]([a-zA-Z]|[0-9])*
number: [0-9]+(.[0-9]+)?(E[+-]?[0-9]+)?
[Defining [a-zA-Z] as *letter* and [0-9] as *digit*, and using those in the regular expressions above is fine.]
(b) [accepting states are marked with double circle]



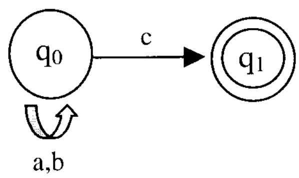
Question 3:

[new computed example]

(a) Thompson's construction of the NFA:



(b) Resulting DFA:



Steps:

(1) Calculate start-state of DFA: ε-closure of state 0: {0,1,2,3,4,5,6,7} → state q_0

(2) Calculate move(q_0, a), move(q_0,b), move(q_0,c)

Move(q_0, a) = {3}

ε-closure({3}) = {0,1,2,3,4,5,6,7} = q0 → no new state added

Move(q_0, b) = {5}

ε-closure({5}) = {0,1,2,3,4,5,6,7} = q0 → no new state added

move(q_0,c) = {8}

ε-closure({8}) = {8} → new state q_1

(3) Calculate move(q1,a), move(q1,b), move(q1,c): all {}. No more states added

(4) Final state for DFA: any new state containing final states of the NFA → q_1

Question 4

[bookwork]

(a) LR parsing involves the use of a parsing table (containing *goto* and *action* entries), and a stack. Given an input string w, the algorithm proceeds as follows:

```
Set input pointer ip to the first symbol of w$
Repeat:
    Let s be the state on top of the stack, and a the symbol pointed by ip
    if action[s,a] = shift s'   then
        begin
            Push a then s' on top of the stack
            Advance ip to the next input symbol
        end
    else if action[s,a] = reduce A->β then begin
            Pop 2*length(β) items off the stack
            Let s' be the state now on top of the stack
            Push A then goto[s', A] on top of the stack
            Output the production A->β
            end
    else if action[s,a]=accept then return
    else error()
    end
```

(b) By constructing the *interference graph*, where :

1. Variables are nodes in the graph
2. An arc drawn between two nodes indicates that the two nodes cannot share a register, because they are live at the same time.

we map the problem of register allocation to the graph colouring problem in graph theory: how to colour the nodes of a graph with the lowest possible number of colours, such that for each arc the nodes at its ends have different colours.

Heuristic algorithm for determining whether a graph is K-colourable:

For each node n in the graph G that has fewer than k-neighbours, we remove n along with its edges. This results in a graph G' and the problem has been reduced to k-colouring of G' (since G can be coloured by assigning to n one of the colours note assigned to any of its neighbours).
Process is repeated until you get either:

- An empty graph (which means that k-colouring of G is possible)
- A graph in which each node has k or more adjacent nodes (which means that k-colouring may not be possible, and spilling code may be needed).