UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2003

MSci Honours Degree in Mathematics and Computer Science Part IV
MEng Honours Degrees in Computing Part IV
MSc in Advanced Computing
PhD
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute
This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER C470

PROGRAM ANALYSIS

Friday 16 May 2003, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1a Consider the following simple functional programming language:

$$
\begin{aligned}
e &::= t^{\ell} \\
t &::= c \mid x \mid \texttt{fn}\ x\ \texttt{=>}\ e_0 \mid e_1\ e_2 \mid \\
&\quad\ \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 \mid e_1\ op\ e_2
\end{aligned}
$$

Write down a syntax-based specification of a control flow analysis for this language (0-CFA).

b Consider the following program:

```
let  d = fn x  =>
                  fn z => if z then (fn w =>z) else x
in   d (fn y => y)
```

Label the program, write down the constraints for a 0-CFA and solve them.

c The control flow analysis is to be extended with a data flow analysis that tracks numerical values using:

$$\mathbf{Range} = \{\texttt{<-1}, \texttt{-1}, \texttt{0}, \texttt{+1}, \texttt{>+1}\}$$

where the elements have the obvious meanings. Write down a suitable definition for $+$, and write down the rules for the analysis (you need only write down the rules that are different from the 0-CFA rules).

(The three parts carry, respectively, 25%, 40%, and 35% of the marks).

2a Consider the following constraint system:

$$\begin{array}{rcl} x_1 & = & A \\ x_2 & = & x_1 \cup x_6 \\ x_3 & = & x_2 \cup x_4 \\ x_4 & = & x_3 \cup B \\ x_5 & = & x_3 \\ x_6 & = & x_5 \cup C \\ x_7 & = & x_1 \cup x_8 \\ x_8 & = & x_7 \cup D \\ x_9 & = & x_2 \cup x_7 \end{array}$$

where A, B, C and D are constant sets. Write down a solution, showing your working.

b Using the constraints from part a, draw a graph of the constraints, write down an algorithm for constructing a depth-first spanning forest and apply it to your graph. Hence or otherwise, write down a rPOSTORDER ordering of the constraints.

c The following is an abstract worklist algorithm that given a set of constraints, $x_1 \sqsupseteq t_1, \cdots, x_N \sqsupseteq t_N$, computes the least solution.

```
INPUT:     A system $\mathcal{S}$ of constraints: $x_1 \sqsupseteq t_1, \cdots, x_N \sqsupseteq t_N$

OUTPUT:    The least solution: Analysis

METHOD:    Step 1:  Initialisation (of W, Analysis and infl)
                    W := empty;
                    for all $x \sqsupseteq t$ in $\mathcal{S}$ do
                        W := insert(($x \sqsupseteq t$),W)
                        Analysis[$x$] := $\bot$;
                        infl[$x$] := $\emptyset$;
                    for all $x \sqsupseteq t$ in $\mathcal{S}$ do
                        for all $x'$ in $FV(t)$ do
                            infl[$x'$] := infl[$x'$] $\cup$ {$x \sqsupseteq t$};

           Step 2:  Iteration (updating W and Analysis)
                    while W $\neq$ empty do
                      (($x \sqsupseteq t$),W) := extract(W);
                      new := eval($t$,Analysis);
                      if Analysis[$x$] $\not\sqsupseteq$ new then
                          Analysis[$x$] := Analysis[$x$] $\sqcup$ new;
                          for all $x' \sqsupseteq t'$ in infl[$x$] do
                              W := insert(($x' \sqsupseteq t'$),W);

USING:     function eval($t$,Analysis)
           return $[\![t]\!]$(Analysis)

           ...
```

Show how the algorithm can be instantiated to give a LIFO-based version of the worklist algorithm.

(The three parts carry, respectively, 30%, 50%, and 20% of the marks).

3a Write a program in the extended WHILE language (which allows for procedure declarations and calls) with two procedures: `one(val y, var z)` and `two(val z, var x)`, where `one` is calling `two` and such that the context-insensitive Assigned Variable *AV* analysis gives:

$$\begin{aligned} AV(\texttt{one}) &= \{\texttt{x}, \texttt{y}\} \\ AV(\texttt{two}) &= \{\texttt{y}\} \end{aligned}$$

b Justify this by computing *AV* (and in particular *IAV* and *ICP*) of `one` and `two`. Note: You do not need to label the program.

c Consider the following WHILE program:

```
if x := 2
then y := 0
else (while y>0 do y := y-1);
x := 0
```

Write down the equations for the Reaching Definition RD analysis of this program. (Label and state explicitly the *flow* of this program and give the auxiliary functions $kill_{\mathsf{RD}}$ and gen_{RD}.)

(The parts carry, respectively, 30%, 30% and 40% of the marks).

4a Consider the following WHILE program:

```
x := 1;
y := 2;
if z<3
then x := z
else x := y;
y := z
```

Perform a Live Variable LV analysis of this program assuming that y is live at the end of the program (label and state explicitly the *flow* of this program, give the auxiliary functions $kill_{LV}$ and gen_{LV}, formulate equations and give the solutions).

b Design a formal system for dead-code elimination based on the LV analysis: An assignment to a dead (i.e. non-live) variable can be replaced by a `skip` statement. A `skip` cannot be eliminated if it is syntactically necessary as in `if` and `while` constructs. For example, we can use the following simplification rules:

$$\frac{\mathsf{LV} \vdash S_2 \triangleright S_2'}{\mathsf{LV} \vdash \texttt{if } [\texttt{b}]^\ell \texttt{ then skip else } S_2 \triangleright \texttt{if } [\texttt{b}]^\ell \texttt{ then skip else } S_2'}$$

or (assuming that the evaluation of b always terminates):

$$\frac{}{\mathsf{LV} \vdash \texttt{if } [\texttt{b}]^\ell \texttt{ then skip else skip} \triangleright [\texttt{skip}]^\ell}$$

but not

$$\frac{}{\mathsf{LV} \vdash \texttt{if } [\texttt{b}]^\ell \texttt{ then } S_1 \texttt{ else skip} \triangleright \texttt{if } [\texttt{b}]^\ell \texttt{ then } S_1}$$

Formulate simplification rules eliminating dead code and redundant `skip` statements in assignments and for sequential compositions.

c Based on the result of the LV analysis above: How can the program be optimised? What would be necessary to optimise it even further?

(The parts carry, respectively, 55%, 20% and 25% of the marks).