UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2002

MSci Honours Degree in Mathematics and Computer Science Part IV
MEng Honours Degrees in Computing Part IV
MSc in Advanced Computing
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

Thursday 25 April 2002, 14:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1a Consider the following simple functional programming language:

$$\begin{array}{lll} \texttt{t} & ::= & \texttt{fn x => e} \mid \texttt{e}_1\ \texttt{e}_2 \mid \texttt{x} \mid \texttt{n} \\ \texttt{e} & ::= & \texttt{t}^{\ell} \end{array}$$

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

b Write down the function $\mathcal{C}_\star[\![e]\!]$ which generates a set of constraints for the expression e.

c Consider the term:

```
((fn x => (x x))(fn y => (y y)))
```

i) Write down a labelled version of the term.

ii) Write down the constraints generated for the 0-CFA of this term (according to your answer to part b or otherwise).

d Using the constraints from part c, or otherwise, produce an analysis result for the program of part c.

2 Consider the following simple functional programming language:

$$
\begin{array}{rl}
\texttt{t ::=} & \texttt{x} \mid \texttt{c} \mid \texttt{nil} \mid \\
& \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \mid \\
& \texttt{fn x => e} \mid \texttt{fun f x => e} \mid \\
& e_1\ e_2 \mid \\
& \texttt{let x = } e_1 \texttt{ in } e_2 \mid \\
& \texttt{hd e} \mid \texttt{tl e} \mid e_1 \texttt{ : } e_2 \mid \texttt{null e}
\end{array}
$$

where expressions are labelled terms.

a Write down an abstract specification of a 0-CFA for the list processing features of this language (`hd`, `tl`, `nil`, `null` and `:`).

b Using $\mathbf{N}_\perp^\top \times \mathcal{P}(\{\mathrm{tt}, \mathrm{ff}\})$, or some other suitable lattice, construct an instance of a monotone structure that could be useful for giving an estimation of the length of lists that might arise as the result of evaluating an expression.

c Write down a specification of 0-CFA extended with the data flow analysis of part b; you need only give the rules for the list processing features and the `if` expression.

3 Consider the following imperative language with statements:

$$
\begin{array}{rcl}
S & ::= & \texttt{x := a} \\
 & | & \textbf{skip} \\
 & | & S_1 \texttt{ ; } S_2 \\
 & | & \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \\
 & | & \textbf{while } b \textbf{ do } S
\end{array}
$$

The *sign* of a variable x is defined to be:

$$
\text{sign(x)} = \begin{cases} + & \text{if x} > 0 \\ 0 & \text{if x} = 0 \\ - & \text{if x} < 0 \end{cases}
$$

This question asks you to define a Sign Analysis Sign as an instance of the monotone framework (similar to the Constant Propagation Analysis CP).

> For each program point, the Sign Analysis will determine the sign a variable may have, whenever execution reaches that point.

Assume the usual labelling. Take the flow, F, to be $\mathit{flow}(S_\star)$, and the extremal labels, E, to be $\{\mathit{init}(S_\star)\}$.

a Define the lattice of signs **Sign** and the lattice of states $\widehat{\textbf{State}}$ underlying the analysis Sign (in particular the appropriate '$\sqsubseteq$' and '$\sqcup$').

b Define the (abstract) evaluation function:

$$\mathcal{A}_{\mathsf{Sign}} : \textbf{AExp} \to (\widehat{\textbf{State}} \to \textbf{Sign})$$

of arithmetic expressions involving addition and multiplication, including $\widehat{+}$ and $\widehat{\times}$.

c Define the transfer functions (for all labels ℓ):

$$f_\ell(\sigma)^{\mathsf{Sign}} : \widehat{\textbf{State}} \to \widehat{\textbf{State}}.$$

(The three parts carry, respectively 40%, 40% and 20% of the marks).

4 Consider the following imperative language with statements of the form:

$$
\begin{array}{rcl}
S & ::= & \texttt{x := a} \\
 & | & \textbf{skip} \\
 & | & S_1 \ ; \ S_2 \\
 & | & \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \\
 & | & \textbf{choose } S_1 \mid S_2 \mid \ldots \mid S_n \\
 & | & \textbf{combine } S_1 \mid S_2 \mid \ldots \mid S_n \\
 & | & \textbf{while } b \textbf{ do } S
\end{array}
$$

In the **choose** statement only one of the $n \geq 1$ statements S_i is actually selected to be executed. The **combine** executes all of the n statements S_i in some sequence. In both statements the choices are made in a non-deterministic way.

This question asks you to define a Live Variable Analysis LV for this extended language, similar to the one for the simple **while** language.

a Define an appropriate labelling for statements/blocks and give a definition for the flow and reverse flow, (together with init and final).

b Define the equation schemes for LV_{entry} and LV_{exit} and all needed auxiliary functions (with their type).

c Use these schemes to formulate the equations for LV analysis for the following program (introduce labels and formulate equations, but do not solve them):

```
combine (  x := 3    |
           y := 0    );
choose (   x := x-1  |
           y := y+x  );
x := y;
```

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