UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2003

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
MSc in Computing Science
BSc Honours Degree in Mathematics and Computer Science Part II
MSci Honours Degree in Mathematics and Computer Science Part II
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 C242=MC242

ASSURED SOFTWARE

Monday 12 May 2003, 14:30
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1) The following is the abstract syntax of a *Sloth control language*:

$$\begin{array}{lll} a & \in & \textit{Arithmetic Expressions} \\ p & \in & \textit{Program} \\ a & ::= & n \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \\ p & ::= & \textbf{step} \mid \textbf{stay}(a) \mid \textbf{left} \mid \textbf{right} \mid p_1 \,\textbf{;}\, p_2 \end{array}$$

A program written in this language controls a mechanic Sloth that moves around on an infinite board, composed of fields, that are either occupied (by, for example, rocks, or other Sloths) or are accessible.

- Using **step**, it can step one field from its current field in the direction that it is pointing. However, if the **step** would bring the Sloth onto an occupied field, the Sloth will not step; instead, it will randomly choose to turn right or left and step again.
- Using **stay**(a), the Sloth will halt a number of clock cycles, equal to the value represented by a.
- Using **left** and **right**, the direction that the Sloth is pointing at will change.

Many Sloths can move around on the board simultaneously, but you can assume that they are controlled by *one* CPU (so synchronisation is not an issue).

The state of the Sloth is represented by a triple: $\langle pos, dir, \mathbf{F}\rangle$. The first element *pos* gives the Sloth's current location in the field, in Cartesian co-ordinates $\langle x, y\rangle$; the second element *dir* gives the direction the Sloth is heading (either *N*, *E*, *S*, or *W*); the third element $\mathbf{F}$ is a double array of booleans indicating whether any field on the board is occupied or not.

a. Assuming the existence of the function $a \mapsto [\![a]\!]_{\mathcal{A}}\, s$ that defines the semantics of arithmetic expressions, give the *Structural Operational Semantics* of Sloth programs.

b. Give the SOS semantics for the program

$$\textbf{step}\,\textbf{;}\,\textbf{stay}(0+2)\,\textbf{;}\,\textbf{right}\,\textbf{;}\,\textbf{step}\,\textbf{;}\,\textbf{step},$$

on the configuration $\langle\langle 0,0\rangle, N, \mathbf{F}\rangle$, where $\mathbf{F}$ is empty but for the coordinates $(2,1)$; you do not need to justify each step via a derivation, just show the sequence.

c. Is the semantics of the previous part *deterministic*? Explain in English.

d. Are all possible programs *terminating*? Explain in English.

e. The *Sloth Machine* itself has configurations

$$\langle c, e, s \rangle \in \mathit{Code} \times \mathit{Stack} \times \mathit{State},$$

where *Stack* is as can be expected, and

$$\begin{array}{rcl} c & \in & \mathit{Code} \\ i & \in & \mathit{Instruction} \\ c & ::= & i \mid c_1 : c_2 \end{array}$$

With the intention of defining a suitable translation function to translate Sloth control programs into Sloth machine code, specify the *set of instructions*, and give the *intended suitable translation function* to translate Sloth control programs into Sloth machine code. Translate the program

$$\texttt{step};\texttt{stay}(0+2);\texttt{right};\texttt{step};\texttt{step}$$

to Sloth Machine code

The five parts carry 35%, 20%, 15%, 15%, and 15% of the marks, respectively.

2) The abstract syntax for the language `from-to-loop` is given by:

$$\begin{array}{lll} x & \in & \textit{Variables} \\ a & \in & \textit{Arithmetic Expressions} \\ b & \in & \textit{Boolean Expressions} \\ S & \in & \textit{Statements} \end{array}$$

$$\begin{array}{lll} a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \\ b & ::= & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \,\&\, b_2 \\ S & ::= & x := a \mid S_1 \texttt{;} S_2 \mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \\ & & \texttt{skip} \mid \texttt{from}\ a_1\ \texttt{to}\ a_2\ \texttt{do}\ S \end{array}$$

The intention of the '$\texttt{from}\ a_1\ \texttt{to}\ a_2\ \texttt{do}\ S$' is that, first, the values of a_1 and a_2 will be checked. If a_2 is greater than or equal to a_1, then S will executed. After that, the entry test will be repeated, checking now the values of a_2 and $a_1 + 1$. This repeats until the test fails.

a. Assume the existence of the functions $a \mapsto [\![a]\!]_{\mathcal{A}}\, s$ and $b \mapsto [\![b]\!]_{\mathcal{B}}\, s$ that define the semantics of arithmetic and boolean expressions. Define the Structural Operational Semantics for the language `from-to-loop`.

b. What does it mean for two programs to be *semantically equivalent*? Show that

$$\texttt{from}\ a_1\ \texttt{to}\ a_2\ \texttt{do}\ P$$

and

$$\texttt{if}\ (a_1 \leq a_2)\ \texttt{then}\ P; \texttt{from}\ (a_1 + 1)\ \texttt{to}\ a_2\ \texttt{do}\ P\ \texttt{else}\ \texttt{skip}$$

are semantically equivalent.

c. Let (part of) an abstract machine be defined by:

$$\begin{array}{lll} \texttt{inst} & ::= & \textbf{PUSH}-n \mid \textbf{ADD} \mid \textbf{MULT} \mid \textbf{SUB} \mid \textbf{TRUE} \mid \textbf{FALSE} \mid \textbf{EQ} \mid \textbf{LE} \mid \textbf{AND} \mid \textbf{NEG} \mid \\ & & \textbf{FETCH}-x \mid \textbf{STORE}-x \mid \textbf{NOOP} \mid \textbf{BRANCH}(c, c) \\ c & ::= & \texttt{inst} \mid c_1 : c_2 \end{array}$$

where its operational semantics '$\cdot \rhd \cdot$' is defined as in the notes.

Assuming the definition of $[\![a]\!]_{\mathcal{CA}}$ and $[\![b]\!]_{\mathcal{CB}}$ that, respectively, define the translation of *Arithmetic Expressions* and *Boolean Expressions* to *Code*, give the definition for $[\![P]\!]_{\mathcal{CP}}$. Notice that this implies that you will have to extend also the definition of `inst`.

The three parts carry 35%, 30%, and 35% of the marks, respectively.

3) a. Consider the following WHILE program:

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

Perform a Live Variable analysis of this program (label and state explicitly *flow* and $flow^R$ of this program, give the auxiliary functions $kill_{LV}$ and gen_{LV}, formulate equations and give the solutions).

b. Consider a situation where certain information should be disclosed only to certain classes of users. Describe a lattice of security classes which allows for formulating security policies involving:

L local users only,
E external users only.

as well as information which is "top secret" or available to all users.

Consider the following WHILE program:

```
while z>0 do
      if y=0
         then x:=1
         else x:=0
```

Is this well-typed with respect to the security classes above, if

(*i*) : x is top secret, y is for local users only, and z is public to everybody, or

(*ii*) : x is public to everybody, y is for local users only, and z is top secret?

In each of these cases: either determine a type for the program (show how) or explain why it is not well typed.

(The parts carry, respectively 60% and 40%).

4) a. Consider two complete lattices $(L, \sqsubseteq_L, \sqcup_L, \sqcap_L, \perp_L, \perp_L)$ and $(K, \sqsubseteq_L, \sqcup_K, \sqcap_K, \perp_K, \perp_K)$. Show how the cartesian product $L \times K = \{(l, k) \mid l \in L, k \in K\}$ also forms a complete lattice: Define an order relation on $L \times K$ and show that it is a partial order, define $\sqcup_{L\times K}$ and $\sqcap_{L\times K}$, as well as top and bottom for $L \times K$.

b. For all statements S in the WHILE language show: If $\langle S, s\rangle \Rightarrow \langle S', s'\rangle$ then:

$$\mathit{flow}(S) \supseteq \mathit{flow}(S')$$

Note: You can use $\mathit{final}(S) \supseteq \mathit{final}(S')$.

(The parts carry, respectively 40% and 60%).

End of Paper