UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2004

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
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

Tuesday 27 April 2004, 14:30
Duration: 120 minutes

*Answer THREE questions*

Paper contains 4 questions
Calculators not required

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

$$\begin{array}{lcl} P & \in & \text{Programs} \\ n & \in & \text{Numeral} \\ a & \in & \text{Arithmetic Expressions} \end{array}$$

$$\begin{array}{lcl} P & ::= & \mathbf{up} \mid \mathbf{down} \mid \mathbf{move}(a_1, a_2) \mid P_1;\ P_2 \\ a & ::= & n \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \end{array}$$

The snail navigates its way around a two dimensional space. It has a pen which may be up or down; in the latter case it leaves an ink trace of its movements. It can move from its current position to the relative coordinates indicated by the vector $(a_1, a_2)$.

a Define the semantics of arithmetic expressions $a$ in **Snail** and use this to define the Structural Operational Semantics of **Snail** programs $P$.

[Hint: The state $s \in$ State of the snail is represented by a triple $\langle x, y, w \rangle$. The first two elements specify the current position while the third one is a boolean indicating whether the pen is up or down.]

b The abstract machine for **Snail** is based on configurations $\langle c, e, s \rangle$ in Code $\times$ Stack $\times$ State where:

$$\begin{array}{lcl} c & \in & \text{Code} \\ i & \in & \text{Instructions} \end{array}$$

$$\begin{array}{lcl} c & ::= & \epsilon \mid i : c \\ i & ::= & \mathbf{PUSH}\text{-}n \mid \mathbf{ADD} \mid \mathbf{SUB} \mid \mathbf{MULT} \mid \mathbf{UP} \mid \mathbf{DOWN} \mid \mathbf{MOVE} \end{array}$$

Define an abstract machine semantics for **Snail** (you only need to specify single step transitions).

c Define suitable translation functions to compile **Snail** programs into abstract machine code.

Describe formally in which way your implementation is *correct* (you do not need to prove the statement).

*The three parts carry, respectively, 25%, 30%, and 45% of the marks.*

2 The abstract syntax of **miniCP** (constraint programming) is given as follows:

$$\begin{array}{rcl} A & \in & \text{Agents} \\ c & \in & \text{Constraints} \end{array}$$

$$\begin{array}{rcl} A & ::= & \mathbf{stop} \mid \mathbf{tell}(c) \mid \mathbf{ask}(c_1) \rightarrow A_1 \,[]\, \mathbf{ask}(c_2) \rightarrow A_2 \\ c & ::= & c \mid c_1 \sqcup c_2 \mid \mathit{true} \mid \mathit{false} \end{array}$$

The *constraints* $c, d, \ldots$ are elements in an (abstract) lattice $(\mathcal{C}, \sqsubseteq, \sqcup)$ where the top element is denoted by *false* and the bottom element by *true*.

The Operational Semantics is specified using configurations given as tuples $\langle A, d\rangle$, where $A$ is a **miniCP** agent, and $d$ a constraint, i.e. an element in the lattice $\mathcal{C}$. The SOS rules are:

$$\frac{}{\langle \mathbf{tell}(c), d\rangle \Rightarrow \langle \mathbf{stop}, c \sqcup d\rangle}$$

$$\frac{\langle A_1, d\rangle \Rightarrow \langle A_1', d'\rangle}{\langle \mathbf{ask}(c_1) \rightarrow A_1 \,[]\, \mathbf{ask}(c_2) \rightarrow A_2, d\rangle \Rightarrow \langle A_1', d'\rangle} \quad \text{if } c_1 \sqsubseteq d$$

$$\frac{\langle A_2, d\rangle \Rightarrow \langle A_2', d'\rangle}{\langle \mathbf{ask}(c_1) \rightarrow A_1 \,[]\, \mathbf{ask}(c_2) \rightarrow A_2, d\rangle \Rightarrow \langle A_2', d'\rangle} \quad \text{if } c_2 \sqsubseteq d$$

a Extend the syntax with a parallel construct

$$A ::= \ldots \mid A_1 \parallel A_2$$

Complete the semantics for the extended **miniCP** language.

What can you say about termination of **miniCP** programs?

b Show the derivation(s) of the following program in extended **miniCP**,

$$P \equiv \mathbf{tell}(c) \parallel (\mathbf{ask}(\mathit{true}) \rightarrow \mathbf{tell}(d) \,[]\, \mathbf{ask}(d) \rightarrow \mathbf{tell}(e))$$

starting with the initial configuration $\langle P, \mathit{true}\rangle$ assuming

(i) : that $\mathit{true} \sqsubset c \sqsubset d$, and

(ii) : that $\mathit{true} \sqsubset d \sqsubset c$.

c Show that the extended **miniCP** is *monotone*, i.e. that for every (partial) derivation sequence $\langle A, d\rangle \Rightarrow^* \langle A', d'\rangle$ we have $d \sqsubseteq d'$. You can assume that for $\langle A, d\rangle \Rightarrow \langle A_1, d_1\rangle$ the derivation sequence for $\langle A_1, d_1\rangle$ is shorter than for $\langle A, d\rangle$.

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

3 Consider the **While** language as presented in the lecture.

A *parity analysis* (as discussed informally in the lecture) attempts to determine for each program point if variables are odd or even.

a Specify a formal parity analysis for **While** along the lines of the Live Variable and Reaching Definition in the lecture. What information is assigned to each program point? What are the auxiliary functions and how do you specify the data-flow equations.

[Hint: Define an abstract semantics for Arithmetic Expressions which determines the parity of an expression based on the current information at each program point. You have to define in particular the meaning of $\underline{+}$, $\underline{-}$ and $\underline{\times}$.]

b Use the formal analysis you specified in part (a) to analyse the following program:

```
m := 2;
while n > 1 do (
        m := m × n;
        n := n − 1 )
skip
```

Label the program, specify the auxiliary functions, state the equations and guess a solution (or solve the equations).

What result would you expect from analysing the program (you do not need to perform a formal analysis):

```
m := 1;
while n > 1 do (
        m := m × n;
        n := n − 1 )
skip
```

Comment on the main difference between the two programs.

*The two parts carry, respectively, 60%, and 40% of the marks.*

4 Extend the While language with a **repeat** statement,

$$S ::= \ldots \mid \textbf{repeat } S \textbf{ until } b$$

The idea is that the body of the **repeat** statement will be executed at least once, after which $b$ is evaluated in order to decide whether to repeat the execution of the body or to terminate the loop.

a Specify the SOS semantics for this statement

b State the basic rules for security typing for the **while** language (i.e. CONST, VAR, EXPR, VAL, ASSIGN, SKP, SEQ, IF, and WHILE) and introduce a new rule RPT for the new statement.

c Using the extended security type system, analyse the following program:

$$\textbf{repeat } x := x + 1; \textbf{ until } y = 1$$

Explain/comment on the result.

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