UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1999

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 Royal College of Science
Associateship of the City and Guilds of London Institute

PAPER 2.12 / MC 2.12

SEMANTICS – OPERATIONAL

Thursday, May 6th 1999, 2.00 – 3.30

Answer THREE questions

For admin. only:
paper contains 4 questions

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

$$\begin{array}{lcl} a & \in & \textit{Arithmetic Expression} \\ p & \in & \textit{Program} \end{array}$$

$$\begin{array}{lcl} a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \\ p & ::= & \texttt{Leap}(a) \mid \texttt{L} \mid \texttt{R} \mid p_1 \,;\, p_2 \end{array}$$

The Cricket 'leaps' around on a board, composed of fields with coordinates that range from $\langle 0, 0\rangle$ to $\langle 120, 100\rangle$, that are either marked or not. While it leaps around, if it reaches an unmarked field, it will mark it; if it reaches a marked field, it will unmark it. Using $\texttt{Leap}(a)$, it can leap from its current field in the direction that it is pointing; the field it reaches in the leap is a fields away from the current. However, if the `Leap` would bring the Cricket outside the board, the instruction is to be ignored (so if the Cricket is at $\langle 90, 95\rangle$, pointing north, performing `jump` (10) would leave the Cricket at $\langle 90, 95\rangle$).

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

a Give the Natural Semantics of Cricket programs.

b The Cricket Machine has configurations $\langle c, e, s\rangle \in \textit{Code} \times \textit{Stack} \times \textit{State}$, where

$$\begin{array}{lcl} c & \in & \textit{Code} \\ i & \in & \textit{Instruction} \\ c & ::= & \varepsilon \mid i : c \end{array}$$

With the intention of defining a suitable translation function to translate Cricket control programs into Cricket machine code, specify the set of instructions, and define an operational semantics for the Cricket Machine.

c Given a mapping $\mathcal{CA}$ from *Arithmetic Expression* into code, give the intended suitable translation function to translate Cricket control programs into Cricket machine code.

d Assuming that:

$$\text{if } \langle c_1, e_1, s\rangle \rhd^k \langle c', e', s'\rangle \text{ then } \langle c_1 : c_2, e_1 : e_2, s\rangle \rhd^k \langle c' : c_2, e' : e_2, s'\rangle,$$

and

$$\langle \mathcal{CA}\,[\![a]\!], \epsilon, s\rangle \rhd^* \langle \epsilon, \mathcal{A}\,[\![a]\!]s, s\rangle$$

show that your translation function for programs is correct with respect to the Natural Semantics. Show at least one base case and one inductive case.

The four parts carry 25%, 25%, 20%, and 30% of the marks, respectively.

Turn over . . .

2 The abstract syntax for the language `From-To-Loop` is given by:

$$
\begin{array}{lcl}
x & \in & \textit{Variable} \\
a & \in & \textit{Arithmetic Expression} \\
b & \in & \textit{Boolean Expression} \\
S & \in & \textit{Statement}
\end{array}
$$

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

The intention of the '`from` a_1 `to` a_2 `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 $a \mapsto \mathcal{A}\,[\![a]\!]s$ and $b \mapsto \mathcal{B}\,[\![b]\!]s$ that define the semantics of arithmetic and boolean expressions. Define the Structural Operational Semantics for `From-To-Loop`.

b Formulate what it would mean for the Structural Operational Semantics for `From-To-Loop` to be *terminating*. Will it be possible to show termination for `From-To-Loop`? If yes, give a sketch of how you would prove this property to hold; if not, give an example of non-termination.

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

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

where (part of) its operational semantics is defined by:

$$
\begin{array}{rcll}
\langle \texttt{PUSH}{-}n : c, e, s\rangle & \rhd & \langle c, \underline{n} : e, s\rangle & \\
\langle \texttt{ADD} : c, z_1 : z_2 : e, s\rangle & \rhd & \langle c, z_1 + z_2 : e, s\rangle & \\
\langle \texttt{MULT} : c, z_1 : z_2 : e, s\rangle & \rhd & \langle c, z_1 \times z_2 : e, s\rangle & \\
\langle \texttt{SUB} : c, z_1 : z_2 : e, s\rangle & \rhd & \langle c, z_1 - z_2 : e, s\rangle & \\
\langle \texttt{TRUE} : c, e, s\rangle & \rhd & \langle c, \texttt{tt} : e, s\rangle & \\
\langle \texttt{FALSE} : c, e, s\rangle & \rhd & \langle c, \texttt{ff} : e, s\rangle & \\
\langle \texttt{EQ} : c, z_1 : z_2 : e, s\rangle & \rhd & \langle c, (z_1 \underline{=} z_2) : e, s\rangle & \text{if } z_1, z_2 \in \mathbb{Z} \\
\langle \texttt{LE} : c, z_1 : z_2 : e, s\rangle & \rhd & \langle c, (z_1 \underline{\leq} z_2) : e, s\rangle & \text{if } z_1, z_2 \in \mathbb{Z} \\
\langle \texttt{AND} : c, b_1 : b_2 : e, s\rangle & \rhd & \langle c, (b_1 \;\&\; b_2) : e, s\rangle & \text{if } b_1, b_2 \in \mathbb{T} \\
\langle \texttt{NEG} : c, b : e, s\rangle & \rhd & \langle c, \underline{\neg}\, b : e, s\rangle & \text{if } b \in \mathbb{T} \\
\langle \texttt{FETCH}{-}x : c, e, s\rangle & \rhd & \langle c, (s\, x) : e, s\rangle & \\
\langle \texttt{STORE}{-}x : c, z : e, s\rangle & \rhd & \langle c, e, s[x \mapsto z]\rangle & \\
\langle \texttt{NOOP} : c, e, s\rangle & \rhd & \langle c, e, s\rangle & \\
\langle \texttt{BRANCH}(c_1, c_2) : c, b : e, s\rangle & \rhd & \langle c_1 : c, e, s\rangle & \text{if } b = \texttt{tt} \\
\langle \texttt{BRANCH}(c_1, c_2) : c, b : e, s\rangle & \rhd & \langle c_2 : c, e, s\rangle & \text{if } b = \texttt{ff}
\end{array}
$$

Turn over ...

Assuming the definition of $\mathcal{CA}$ and $\mathcal{CB}$ that, respectively, define the translation of *Arithmetic Expression* and *Boolean Expression* to *Code*, let the following be (part of) the definition of programs in `From-To-Loop` to *Code*:

$$\begin{aligned}
\mathcal{CS}\,[\![x := a]\!] &= \mathcal{CA}\,[\![a]\!] : \texttt{STORE}{-}x \\
\mathcal{CS}\,[\![\texttt{skip}]\!] &= \texttt{NOOP} \\
\mathcal{CS}\,[\![S_1\,;\,S_2]\!] &= \mathcal{CS}\,[\![S_1]\!] : \mathcal{CS}\,[\![S_2]\!] \\
\mathcal{CS}\,[\![\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2]\!] &= \mathcal{CB}\,[\![b]\!] : \texttt{BRANCH}(\mathcal{CS}\,[\![S_1]\!], \mathcal{CS}\,[\![S_2]\!])
\end{aligned}$$

Extend the definition with a case for $\mathcal{CS}\,[\![\texttt{from}\ a_1\ \texttt{to}\ a_2\ \texttt{do}\ S]\!]$. Notice that this implies that you will have to extend also the definition of both `inst` and $\triangleright$.

d Show that the above defined translation function is correct with respect to Structural Operational Semantics. The proof will follow an inductive reasoning; you only need to show the case that deals with '$\texttt{from}\ a_1\ \texttt{to}\ a_2\ \texttt{do}\ S$' and assume all other cases to be proven correct.

Turn over . . .

3 The abstract syntax for the language `While` is given by:

$$
\begin{array}{lcl}
x & \in & \textit{Variable} \\
a & \in & \textit{Arithmetic Expression} \\
b & \in & \textit{Boolean Expression} \\
S & \in & \textit{Statement}
\end{array}
$$

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

a Briefly describe each of the three main approaches to specifying programming language semantics: Natural, Structural Operational, and Denotational Semantics. Identify their comparative strengths and weaknesses.

b Assuming the existence of suitable functions $a \mapsto \mathcal{A}\,[\![a]\!]s$ and $b \mapsto \mathcal{B}\,[\![b]\!]s$ for the semantics of both arithmetic and boolean expressions, give the definition of Natural Semantics and Structural Operational Semantics for `While`.

c Assuming that:

$$\text{if } \langle S_1, s_1\rangle \to s_2 \text{ then } \langle S_1\,;\,S_2, s_1\rangle \to \langle S_2, s_2\rangle,$$

show that, for the language `While`, Structural Operational and Natural Semantics coincide. The proof will follow an inductive reasoning. From left to right, it suffices to show the result for assignment and the conditional; in the other direction, focus on (while_{sos}).

d The While language is extended with parameterless procedures. In particular, there is a new Calling statement:

$$S ::= \ldots \mid \texttt{call}\ p$$

i. Assuming that the language has dynamic scope rules, write down the Natural Semantics for the `call` statement.

ii. Assuming that the language has static scoping for procedures (but not variables), write down two alternative Natural Semantics for the `call` statement.

The four parts carry, respectively, 15%, 15%, 40%, and 30% of the marks.

Turn over ...

4 a Consider the space of partial functions from *State* to *State*.

i. Give the definition of a *partial ordered* set.
ii. How is the partial order relation $\sqsubseteq$ defined on functions?
iii. Give the definition of a *least upper bound.*
iv. Give the definition of a *chain.*
v. Give the definition of a *ccpo.*
vi. When is a function *monotone*?
vii. What is a *continuous* function?

b Let $f : D \to D$ be a continuous function on the ccpo $(D, \sqsubseteq)$ and let $d = \sqcup\{f^n \perp \mid n \geq 0\}$.

i. When is w a *fixed point* of f? (NB: you are not to use $\sqcup$ here!)
ii. Show that d is a fixed-point of f.
iii. Show that d is the least fixed-point of f.

c The abstract syntax for the language `Repeat` is given by:

$$\begin{array}{lcl} x & \in & \textit{Variable} \\ a & \in & \textit{Arithmetic Expression} \\ b & \in & \textit{Boolean Expression} \\ S & \in & \textit{Statement} \end{array}$$

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

Assuming the existence of suitable functions $a \mapsto \mathcal{A}\,[\![a]\!]s$ and $b \mapsto \mathcal{B}\,[\![b]\!]s$ for the semantics of both arithmetic and boolean expressions, the denotational semantics of statements is defined by:

$$\begin{array}{rcl} \mathcal{DS}\,[\![x := a]\!]s & = & s[x \mapsto \mathcal{A}\,[\![a]\!]s] \\ \mathcal{DS}\,[\![\texttt{skip}]\!] & = & id \\ \mathcal{DS}\,[\![S_1 \,;\, S_2]\!] & = & \mathcal{DS}\,[\![S_1]\!] \circ \mathcal{DS}\,[\![S_2]\!] \\ \mathcal{DS}\,[\![\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2]\!] & = & \texttt{cond}(\mathcal{B}\,[\![b]\!], \mathcal{DS}\,[\![S_1]\!], \mathcal{DS}\,[\![S_2]\!]) \\ \mathcal{DS}\,[\![\texttt{repeat}\ S\ \texttt{until}\ b]\!] & = & \texttt{FIX}\ F, \text{ where } Fg = \texttt{cond}(\mathcal{B}\,[\![b]\!], id, g) \circ \mathcal{DS}\,[\![S]\!] \end{array}$$

Calculate the denotational semantics for

$$\texttt{if}\ x \leq 0\ \texttt{then}\ x := 5\ \texttt{else}\ \texttt{repeat}\ x := x - 1\ \texttt{until}\ x < 0$$

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

End of paper