UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2000

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BEng Honours Degree in Mathematics and Computer Science Part II
MEng 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

SEMANTICS - OPERATIONAL

Tuesday 16 May 2000, 14:00
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions

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

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

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

A program written in this language controls a mechanic Turtle that moves around on an infinite board, composed of fields, that are either occupied (by, for example, stones) 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 Turtle onto an occupied field, the Turtle will not step; instead, it will randomly choose to turn right or left and step again.
- Using `Stay`(a), the Turtle will halt a number of clock cycles, equal to the value represented by a.
- Using `L` and `R`, the direction that the Turtle is pointing at will change.

Many turtles can move around on the board simultaneously, but you can assume that they are controlled by *one* CPU.

The state of the Turtle is represented by a triple: $\langle pos, dir, \texttt{F}\rangle$. The first element *pos* gives the Turtle's current location in the field, in Cartesian co-ordinates $\langle x, y\rangle$; the second element *dir* gives the direction the Turtle 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 occupied or not.

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

b Is the semantics of the previous part *deterministic*? If yes, give a proof; if no, explain in English.

c The *Turtle Machine* itself has configurations

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

where *Stack* is as can be expected, and

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

With the intention of defining a suitable translation function to translate Turtle control programs into Turtle machine code, specify the *set of instructions*, and give the *intended suitable translation function* to translate Turtle control programs into Turtle machine code. (Notice that your answer to the previous part plays a role here.)

d Using the answer you have given in the previous part, define an *operational semantics* '$\cdot \rhd \cdot$' for Turtle Machine code.

The four parts carry 35%, 15%, 20%, and 30% of the marks, respectively.

2 The abstract syntax for the language `Repeat-Times` is given by:

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

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

The intention of the '`repeat` S a `times`' is that, first, the value of a will be checked. If a is greater than or equal to 1, then S will be executed. After that, the entry test will be repeated, checking now the value of $a - 1$. 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 *Natural Semantics* for `Repeat-Times`.

b Are the programs

$$\texttt{repeat}\ S\ a\ \texttt{times}$$

and

$$\texttt{if}\ a > 0\ \texttt{then}\ (S; \texttt{repeat}\ S\ (a-1)\ \texttt{times})\ \texttt{else skip}$$

semantically equivalent? If yes, give a proof; if no, explain in English.

c Again assuming 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 `Repeat-Times`.

d Show that the Natural Semantics and the Structural Operational Semantics for `Repeat-Times` *coincide*. Your proof should at least deal with '$x := a$' and '`repeat` S a `times`'; you can assume that $\langle S_1, s_1\rangle \Rightarrow^* s_2$ and $\langle S_2, s_2\rangle \Rightarrow^* s_3$ imply $\langle S_1 \; ; \; S_2, s_1\rangle \Rightarrow^* s_3$.

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

3 Consider the following abstract syntax:

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

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

The intention of the 'S_1 `or` S_2' construct is that either S_1 or S_2 will be executed, after a non-deterministic choice; that of '`if` b S' is that S only gets executed if the boolean is true.

a 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 this language; you can restrict your definition to the cases dealing with '`if` b S' and 'S_1 `or` S_2'.

b Are these two semantics *equivalent*? Motivate your answer in English, no formal proof for either answer is required.

c Consider the programming language that has the traditional conditional construct '`if` b `then` S_1 `else` S_2' instead of '`if` b S'. Does that languages have the *same expressive power* as the one defined above? I.e, is it possible to define a translation from one to the other, and vice versa? Motivate your answer, no formal proof is required.

d Extend the syntax above such that it will have *block structures* as well as *parameterless procedures*. (You just need to give the extensions, it is not necessary to repeat the constructs that are already given above.)
Assuming that the language has dynamic scope rules, for both variables and procedure names, write down the Natural Semantics for the '`call`' statement and for the '`begin-end`' construct.

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

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

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

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

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, give the *Denotational semantics* of statements.

c *Calculate* the denotational semantics for

$$\texttt{if}\ x \geq 0\ \texttt{then}\ (\texttt{repeat}\ x := x{-}1\ \texttt{until}\ x = 0)\ \texttt{else}\ x := 5$$

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

End of Paper