UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2002

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing Part I
BSc Honours Degree in Mathematics and Computer Science Part I
MSci Honours Degree in Mathematics and Computer Science Part I
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 C141=MC141

REASONING ABOUT PROGRAMS

Monday 29 April 2002, 16:00
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions
Calculators not required

1 This question is about the Java method firstOccurs below, in which the elements of the array a, the parameter of firstOccurs, are to be interpreted as an integer in binary notation. The post-condition states that the result **r** is the index of the most significant digit of this integer.

```
int firstOccurs(int [ ] a) {
 //pre: a.length > 0 &∀i:int(0 ≤ i < a.length → (a[i] = 0 or a[i] =1))
 //      (i.e. a is a non-empty array of binary digits)
 //post: a = a0 &(r = a.length-1 & ∀j:int(0 ≤ j < a.length → a[j] = 0)) or
 //             (a[r] =1 & 0 ≤ r < a.length & ∀j:int(0 ≤j < r → a[j] = 0)),
 //              where r is the result
 int k = 0;
 //invariant is true here: (part b)
 //variant: a.length - k - 1
 while ((a[k] == 0) && (k < a.length - 1))   k++;
 return k;   // invariant true, while condition false, post-condition true
 }
```

a) State the expected result of firstOccurs for the arrays:

(i)

0	0	1	0	1	1	0

representing 22

(ii)

0	0	0	0	0	0	0

representing 0

(iii)

0	0	0	0	0	0	1

representing 1

b i) Draw a diagram detailing the state of the computation at the start of an arbitrary iteration of the while loop in terms of the relevant variables.

ii) Write down a suitable invariant from the diagram.

c Show carefully that the code re-establishes the invariant.

d Show carefully that the post-condition is achieved after the return.

The four parts carry, respectively, 10%, 25%, 30% , 35% of the marks.

2 a Using double (course of values) induction, or otherwise, show that $\forall$x:Nat ($\forall$y:Nat `fun x y = y+1`), where

```
fun :: Int -> Int -> Int
--pre: x≥0 & y≥0
--post:r = y+1, where r=fun x y
fun 0 y = y+1
fun x y
    | (x>0 && y==0) = (fun (x-1) 1)-1
    | (x>0 && y>0)  = (fun (x-1) (fun x (y-1)))
```

b Given

```
insert :: Int -> [Int] -> [Int]
--pre: order(xs)
--post:perm(r,(x:xs)) & order(r),
--              where r=insert x xs
insert x [] = [x]
insert x (y:ys)
    | x <= y      = (x:(y:ys))
    | otherwise   = (y:(insert x ys))
```

show by list induction on xs that `insert` satisfies its specification. i.e. show

$\forall$xs:[Int]($\forall$c:Int(order(xs) $\rightarrow$ perm(r, (c:xs)) & order(r)))
where r=`insert c xs`.

You should state any facts you use about ordered lists and permutations of lists.
e.g. if order((x:xt)) then order(xt);
if perm((C:xt), y) then perm((C:(x:xt)), (x:y)).

Assume that *perm(u,v)* holds when *u* is a permutation of the elements in *v* and *order(u)* holds when the elements of *u* are in ascending order.

The two parts carry, respectively, 50%, 50% of the marks.

3 Consider the tail-recursive Haskell function `trdivide`, that divides the concatenation of its arguments (`xs`, `us`, `vs`) into two lists: `ys`, consisting of the True elements, and `zs`, consisting of the False elements.

```
trdivide::[Bool]->[Bool]->[Bool]->([Bool],[Bool])
--pre: ∀k[in(k,us) →k=True)&∀k[in(k,vs) →k=False)
--post:perm(xs++us++vs,ys++zs)&
--     ∀k[in(k,ys) →k=True)& ∀k[in(k,zs) →k=False),
--     where (ys,zs)=trdivide xs us vs
trdivide [] us vs  = (us,vs)
trdivide (x:xs) us vs
      | (x==True)   = trdivide xs (x:us) vs
      | (x==False)  = trdivide (rot xs) us (x:vs)
```

where the function `rot::[Bool]->[Bool]` returns the list that results when the last element of its argument `xs` is moved to the front of `xs`.
e.g. `rot [False,False,True] = [True, False, False]`.

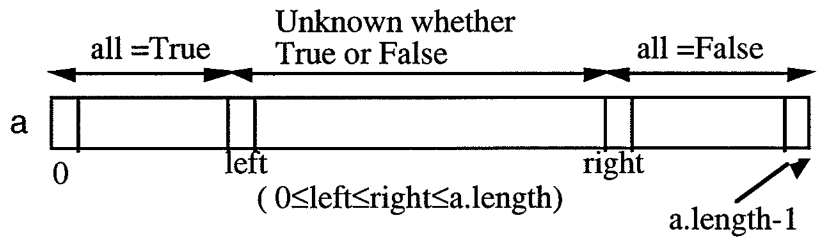
Assume that *perm(u,v)* holds when *u* is a permutation of the elements in *v* and *in(w,u)* holds when *w* is an element of *u*.

The Java method jDivide is to implement `trdivide xs [] []`, where xs is a boolean array a, by swapping elements of a. Its specification uses arrays-as-lists notation, where a(i to j), i≤j, is the list [a[i], ..., a[j-1]]:

```
int jdivide(boolean [ ] a) {
  //pre: none
  //post: a(0 to r) = ys & a(r to a.length) = zs,
  //          where (ys,zs) = trdivide a0 [ ] [ ]
  int left = 0, right = a.length;
  while <while condition>  (part (c))
  //invariant :          (part (b))
  //variant: right - left
    {<loop code>}  (part(c))
  return left; }
```

a Describe in English the result of evaluating `trdivide xs [] []`.

b Use the diagram below, which shows the computation state of an arbitrary iteration of the while loop of jDivide, to give a suitable invariant for jDivide in terms of `trdivide`, left and right.



(**Hint:** arrays-as-list notation is helpful, as is the form of the post-condition.)

c Give the while condition and loop code of jDivide.

d Show that the invariant of jDivide is established *before* the start of the loop code *and* that the post-condition is established at the return.

The four parts carry, respectively, 10%, 25%, 30%, 35% of the marks.

4 a The following code implements Warshall's algorithm.

```
boolean [ ] [ ] warshall(boolean [ ] [ ] a) {
  //pre: none
  //post: result = p &
  //       p[ i ][ j ] is true <-> there is a path in a between i and j.
 boolean [ ] [ ] p = (boolean [ ] [ ]) a.clone();  // copies a into p
 int n = 0;
  while (n<p.length) {
    //invariant : (to be filled in)
    //variant: p.length - n
    for (i = 0; i < p.length; i++)
       for (j = 0; j < p.length; j++)
         p[ i ][ j ] = p[ i ][ j ] || (p[ i ][ n ] && p[ n ][ j ]);
    n++;
    }
  return p;
  }
```

i) State the invariant of the while loop of Warshall's algorithm.

ii) Explain why the post-condition is achieved at the end of the method.

iii) Show carefully that the invariant is re-established by the while-loop code.

b Consider the code below, that is supposed to swap the values of x and y.

```
             //x=x0 & y=y0
   x = x+y;  // (1)
   y = x-y;  // (2)
   x = x-y;  // (3)
```

State the post-condition at (3) and show carefully, by reasoning using mid-conditions, that the given code achieves the post-condition.

Parts ai, aii, aiii, b carry, respectively, 15%, 15%, 35%, 35% of the marks.