UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1998

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

PAPER 1.8

HASKELL AND PROLOG

Friday, May 8th 1998, 2.30 - 4.00

Answer THREE questions

For admin. only: paper contains 4 questions

Section A - Haskell Programming *(Use a separate answer book for this Section)*

1a Write a function `conv`, which takes as input a non-negative integer and returns as output a string representation of the number. So

```
conv 4 = "4"
conv 57 = "57"
```

You may use the built in functions `chr` and `ord`. The function `chr` takes a number `n` and returns the character whose ascii value is that number. So

```
chr 48 = '0'
```

The function `ord` takes a character and returns its ascii value. So

```
ord '0' = 48
```

b Given

```
data Coin = OneP|TwoP|FiveP|TenP
                deriving (Eq, Ord, Show)
type Money = [(Coin, Int)]
```

`Coin` names the following denominations: 1p, 2p, 5p, 10p. `Money` holds the number of coins one has. So 3 five pence pieces and 4 ten pence pieces is `[(FiveP,3),(TenP,4)]`.

i) Write a function `convert` which takes as input something of type `Coin` and returns as output its value. So

```
convert TenP = 10
```

ii) Write a function `value` which takes as input something of type `Money` and returns as output the total value of the coins. You may call `convert` in `value` if you need it. So

```
value  [(FiveP,3),(TenP,4)] = 55
```

iii) Write a function `put` which takes as input a `Coin` and returns as output a string representation of the `Coin`. So

```
put TenP = "ten pence"
```

iv) Write a function `costs` which takes as input the price of an item in pence and returns as output something of type `Money` containing the fewest coins needed to purchase the item. So

```
costs 79 = [(TenP,7),(FiveP,1),(TwoP,2)] .
```

2 a We can represent a set as an ordered list of elements without repetitions (ascending order):

```
type Set = [Int]
```

Define:

```
memSet :: Set -> Int -> Bool
```

which tests to see whether the integer is a member of the set, and:

```
union :: Set -> Set -> Set
```

which produces the union of two sets.

b Assuming the existence of

```
sort :: [Int] -> [Int]
```

which sorts a list of integers into ascending order, define:

```
makeSet :: [Int] -> Set
```

which converts a list to a set.

c Using your answer to part b, define:

```
mapSet :: (Int -> Int) -> Set -> Set
```

which applies its first argument to each element of its second argument.

d A binary relation on integers may be represented as an ordered list of pairs of integers without repetitions:

```
type Relation = [(Int,Int)]
```

Define:

```
image :: Relation -> Int -> Set
```

which produces the image in the relation of the second argument (i.e. a set of values which appear as the second component of a pair in the relation whose first component is the given integer).

Turn Over ...

SECTION B: Prolog Programming
(Use a separate answer book for this section)

In this section you may freely use any of Prolog's built-in relations.

3 On one side of a river there stands a farmer with a wolf, a goat and a cabbage. He wants all of these and himself to be on the other side. He can cross the river in a boat, either alone or accompanied by just one other item. However, it is not *safe* to leave behind the wolf with the goat, as the wolf might eat the goat. Likewise it is not *safe* to leave behind the goat with the cabbage. To achieve his goal the farmer will therefore have to make several crossings back and forth.

a Write two rules which jointly define

crossing(Items, [B1, B2], [A1, A2])

representing a single crossing of the river. Items is a list of those items, including the farmer, that make the crossing. Use f, w, g and c to denote items. B1 and B2 are lists of the items on the two sides of the river before this crossing. A1 and A2 are lists of the items on the two sides of the river after this crossing.

For instance, the predicate crossing([f, g], [[f, w, g, c], []], [[w, c], [f, g]]) expresses that all four items were on one side and the farmer then crossed the river taking the goat with him.

Your two rules may refer to the relations

safe(A1) expressing that it is *safe* to leave behind the items in A1;

remove_all(Items, B1, A1) expressing that A1 is the result of removing from B1 all those items, if any, that occur in Items.

[Hint—one rule is for the farmer crossing alone, and the other rule is for him taking any single item with him.]

b Write one or more rules sufficient to define safe(A1).

c The farmer's desired goal can be pursued using two recursive rules each of the form

state(B1, B2) :- ..., state(A1, A2).

and posing the query ?-state([f, w, g, c], []). Complete the bodies of the two rules and add the single base case needed to express the goal state.

[Hint—one rule is for the farmer being in B1, and the other rule is for him being in B2.]

d Indicate in words how your rules in part c could, in principle, be elaborated so as to prevent execution from generating an infinite computation.

The four parts carry, respectively, 40%, 10%, 40% and 10% of the marks.

4 Assume that the following built-in relations are available, besides any others you may wish to use:

var(X)	"X is a variable" —i.e. X is currently unbound
atomic(X)	"X is either an atom or a number" —e.g. X could be abc or 123
X=..Y	"Y=[P | Args] when X is a compound term P(Args)" —e.g. the call f(a, Z)=..Y will bind Y to [f, a, Z]
forall(X, Y)	"for all ways in which X is true, so also is Y"
findall(X, P, Xs)	"Xs is a list of all values of X for which P is true"

a Write a program for the relation isground(X) which holds only when X is a ground term (i.e. is neither a variable nor contains any variables).

Indicate how it would work for the query ?-isground(f(a, g(2, c))).

b Write a program for the relation atomics(X, Ats) which holds only when

X is a variable and Ats=[]
or X is atomic and Ats=[X]
or X is compound and Ats is a list of all the atomic terms occurring in X.

For instance, the query ?-atomics(f(b, g(Z, 1)), Ats) has a solution Ats=[1, b].

Note—Ats may be in any order, may contain duplicates and ignores functors (such as f and g in the example just given).

c Write a program for the relation hasvar(X, V) which holds only when

either X is a variable and V=X
or X is a compound term containing the variable V.

For instance, the query ?-hasvar(f(Y, g(Z, c)), V) has two solutions V=Y and V=Z.

d Write a program to solve the query ?subsvars(X, 1, Nums) which binds all the distinct variables occurring in X (which may be any term) to distinct positive integers, and returns these integers in the list Nums.

For instance, the query ?-subsvars(f(Y, g(Z, c, h(Y))), 1, Nums) would return Nums=[1, 2] having bound Y to 1 and Z to 2.

Hint—you need one base case and one recursive rule whose head is subsvars(X, K, [K \| Ks]) and which uses hasvar to find a variable occurring in X.

The four parts carry, respectively, 25%, 30%, 15% and 30% of the marks.

End of Paper