UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1998

BEng Honours Degree in Computing Part III
BEng Honours Degree in Information Systems Engineering Part III
MEng Honours Degree in Information Systems Engineering Part III
BSc Honours Degree in Mathematics and Computer Science Part III
MSci Honours Degree in Mathematics and Computer Science Part III
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 Associateship of the Royal College of Science

PAPER 3.27 / I3.10

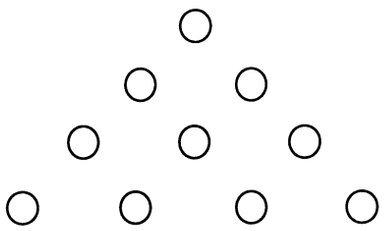
THE PRACTICE OF LOGIC PROGRAMMING

Friday, May 1st 1998, 2.30 - 4.30

Answer THREE questions

For admin. only: paper contains 4 questions

1 Slovak Checkers is played on a wooden board with ten holes, set out in the following triangular pattern:



At the start, there is a peg in every hole. The player removes an arbitrary peg and thereafter repeatedly hops one peg over an adjacent one into a vacant hole, immediately beyond - removing the peg that was jumped over from the board. A jump must always be in a straight line (horizontally or diagonally) and only one peg can be hopped over at a time. The aim is to end up with a single peg on the board.

a Explain how the states of the game can be represented in Prolog, and express in Prolog:
`initial(State)` - a state ***after*** removal of the first arbitrary peg, and
`goal(State)` - a state in which only one peg is left on the board.

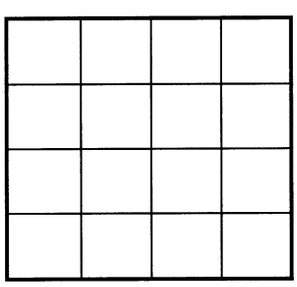
b Represent in Prolog the information about which sets of three adjacent holes are in line.

c Define in Prolog: `move(State1, State2)`, meaning that State2 results from a valid jump in State1.

d Define in Prolog: `solution(L)`, meaning that L is a sequence of states, starting with an initial state and, via a succession of valid jumps, ending up in a goal state.

The four parts carry, respectively, 20%, 10%, 50%, 20% of the marks.

2 The Czech Coin Puzzle involves placing ten gold Crowns onto cells of a four by four grid:



in such a way that there are an even number of coins on each row, column and both of the two principal diagonals (i.e., the ones leading from top-left to bottom-right and from top-right to bottom-left).

a Show how the puzzle can be solved in Prolog. You need not be concerned with efficiency.

b i) Give a more efficient program for finding a solution, using the facilities of a Constraint Logic Programming language over Finite Domains, such as CLP(FD).

ii) Briefly discuss the role of the various sections of your CLP(FD) program and explain how the strategies of the Prolog and CLP(FD) programs differ.

YOU ARE NOT REQUIRED TO FIND A PARTICULAR SOLUTION YOURSELF !!!

The three subparts carry, respectively, 40%, 30%, 30% of the marks.

3a Write the top level `demo` code for a Prolog interpreter that can handle the matching of goals to the heads of clauses, conjunctions of goals, and negation as failure. Your interpreter need **NOT** cater for arithmetic, findall, cut or other built-in primitives, nor need it be concerned with user-interaction nor explanations.

b When applied to the following knowledge base:

```
loves(chris, newts).
loves(chris, prolog).
loves(frank, X) :-
      loves(chris, X),
      not amphib(X).

amphib(newts).
```

a simple Prolog trace program, based on such an interpreter, produces the following output:

```
?- demo(loves(frank,X)).

trying goal: loves(frank,_00A2)
     found matching clause: loves(frank,_00A2) :- loves(chris,_00A2),not amphib(_00A2)
     trying first condition of: loves(chris,_00A2),not amphib(_00A2)
          found matching fact: loves(chris,newts)
     trying second condition: not amphib(newts)
          found matching fact: amphib(newts)
     goal: not amphib(newts) FAILS
     trying first condition again
          found matching fact: loves(chris,prolog)
     trying second condition: not amphib(prolog)
          goal: amphib(prolog) FAILS - no (more) matching clauses
     goal: not amphib(prolog) SUCCEEDS
     conditions: loves(chris,prolog),not amphib(prolog) SUCCEED
goal: loves(frank,prolog) SUCCEEDS

X = prolog
```

(The _00A2 in the above trace is just the internal name for a Prolog variable).

i) Modify the interpreter you wrote in part a, so that it can produce a similar trace, but **without** the (pretty-printing) indentation. Include any ancilliary code for generating the trace messages and make sure that the tracer operates correctly in case of failure and backtracking, as in the example.

ii) Further modify your program so that trace messages are indented, in the manner shown in the example.

The three subparts carry, respectively, 20%, 50%, 30% of the marks.

Turn over ...

4 A matrix can be represented in Prolog and in Constraint Logic Programming languages as a list of rows, each of which is itself a list. For example, the matrix:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 4 & 9 \\ 1 & 8 & 27 \end{pmatrix}$$

can be represented by the list: `[[1,2,3],[1,4,9],[1,8,27]]`.

a Give CLP(R) clauses together with any ancilliary code for:
`row(I, M, R)`, meaning that R is a list representing the Ith row of matrix M, and
`column(J, M, C)`, meaning that C is a list representing the Jth column of matrix M.

b Matrix A can be multiplied by matrix B, i.e., they are *conformable*, provided that the number of columns of A equals the number of rows of B, in which case the ik^{th} entry of the product is the *inner product* of the i^{th} row of A with the k^{th} column of B. Thus, if A is an m x n matrix $[a_{ij}]$ and B is an n x p matrix $[b_{jk}]$, then AB is the m x p matrix, of which the ik^{th} entry is:

$$\sum_{j=1}^{n} a_{ij} b_{jk}$$

Give CLP(R) clauses for `inner_prod(I, J, M1, M2, IP)`, meaning that IP is the inner product of the Ith row of M1 with the Jth column of M2, where M1 and M2 are conformable matrices.

c A straightforward, albeit sometimes relatively inefficient, method of finding the inverse M2 of an invertible square matrix M1 is to set M2 to be isomorphic to M1 (i.e., with the same number of rows and columns) but with variable elements, multiply M1 by M2, and solve the set of equations obtained by equating the entries of the product matrix to 1, where the row and column indexes are the same, and to 0 otherwise.

i) Give the CLP(R) clauses for `constrain_element(I, J, M1, M2)` that embody the appropriate equation for the IJ^{th} entry of the product of matrices M1 and M2, as described in the previous paragraph.

ii) Give CLP(R) clauses for `constrain_row(I, M1, M2)` that embody the appropriate equations for the whole I^{th} row of the product of matrices M1 and M2.

iii) Give CLP(R) clauses for `constrain(M1, M2)` that that embody the whole set of equations for the product of matrices M1 and M2.

iv) Give CLP(R) clauses for `inverse(M1, M2)`, meaning that matrix M2 is the inverse of square matrix M1. Include any ancilliary clauses not already defined.

YOU MAY USE THE SYNTAX OF THE CLP(R) MODULE OF SICSTUS PROLOG OR OF THE ORIGINAL IBM/MONASH CLP(R).

The six subparts carry, respectively, 20%, 10%, 20%, 20%, 10%, 20% of the marks.

End of paper