UNIVERSITY OF DUBLIN
TRINITY COLLEGE

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science Programme **Trinity Term 2014**
B.A. (Mod.) CSLL
Mathematics

Symbolic Programming

Sat 17/05/2014 **LUCE UPPER** **9:30-11:30**

Dr Tim Fernando

Instructions to Candidates:

Attempt **two** questions (out of the three given).

All questions carry equal marks. 50 marks per question.

You may not start this examination until you are instructed to do so by the Invigilator.

Materials permitted for this examination:

Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used.

1. (a) The binary predicate `listSum(+List,?Sum)` below adds up the elements of a list, assumed to be numbers.

```
listSum([],0).
listSum([Head|Tail],Sum) :- listSum(Tail,TailSum),
                            Sum is Head + TailSum.
```

Explain why `listSum` is *not* tail-recursive. Because the recursive function is not called last.

[5 marks]

```
listSum(L,S) :- accLSum(L,0,S).
accLSum([],S,S).
accLSum([H|T],Acc,S) :- NewA is Acc+H, accLSum(T,NewA,S).
```

(b) Modify the predicate in (a) to a tail-recursive predicate that is otherwise equivalent (i.e. true of the same lists and numbers).

[10 marks]

(c) Define a tail-recursive binary predicate `listProd(+List,?Prod)` multiplying the elements of a list, assumed to be numbers. For example,

```
?- listProd([2,3],P).
P= 6 ;
no
```

```
listProd(L,P) :- accListProd(L,1,P).

accListProd([],P,P).
accListProd([H|T], Acc, P) :- NewAcc is H*Acc, accListProd(T, NewAcc, P).
```

[5 marks]

(d) Define a binary predicate `list2N(+Int,?List)` that given a non-negative integer `Int` returns a list in decreasing order of integers from `Int` to 1. For example,

```
?- list2N(0,L).
L=[]  ? ;
no

?- list2N(3,L).
L=[3,2,1]  ? ;
no
```

```
list2N(0,[]).
list2N(X,[X|T]) :- NX is X-1, NX >= 0, list2N(NX, T).
```

[5 marks]

(e) Define a unary predicate `list2N(+List)` that is true when `List` is a non-increasing finite list of numbers. For example,

```
?- nonInc([]).
yes

?- nonInc([7,5,5,0]).
```

```
nonInc([_]).
nonInc([X,Y|T]) :- X > Y, nonInc([Y|T]).
```

```
yes

?- nonInc([7,9,3]).
no
```

[10 marks]

(f) Define a binary predicate `sumList(+Sum,?List)` that given a non-negative integer `Sum` returns a non-increasing list `List` of positive integers that add up to `Sum`. For example,

```
?- sumList(0,L).
L=[]  ? ;
no

?- sumList(3,L).
L = [3] ? ;
L = [2,1] ? ;
L = [1,1,1] ? ;
no
```

[15 marks]

2. (a) Define a binary predicate `subset(L1,L2)` that is true exactly when every member of `L1` is a member of `L2`.

```
subset([],_).
subset([H|T], L2) :- member(H,L2), subset(T,L2).
```

[5 marks]

(b) Define a binary predicate `setEq(L1,L2)` that is true exactly when every member of `L1` is a member of `L2`, and every member of `L2` is a member of `L1`. You are not allowed use to use the built-in predicate `setof`. [5 marks]

```
setEq(L1,L2) :- subset(L1,L2), subset(L2,L1).
```

(c) Use the built-in predicate `setof` to define the same predicate in (b). [5 marks]

```
setEq2(L1,L2) :- setof(C,member(C,L1),S), setof(B,member(B,L2),S).
```

(d) Define a binary predicate `nonmember(X,L)` that is true exactly when `X` is not a member of `L`.

```
nonmember(X, L) :- \+member(X,L).
```

[5 marks]

(e) Define a 3-ary predicate `setInt(List1,List2,ListBoth)` that is true exactly when the members of `ListBoth` are all the members of both `List1` and `List2`. You are not allowed use to use the built-in predicate `setof`.

```
setInt(L1,L2,LBoth) :-
    append(L1,L2,LConcat),
    setEq(LConcat, LBoth).
```

[5 marks]

(f) Use the built-in predicate `setof` to define the same predicate in part (e).

```
setInt2(L1,L2,LBoth) :-
    append(L1,L2,LConcat),
    setEq2(LConcat,LBoth).
```

[5 marks]

(g) Use cut `!` to define the 3-ary predicate `if(A,B,C)` that returns `B` if `A` else `C`.

```
if(A,B,C) :-
    A, !, % if a
    B;    % then b
    C.    % else c
```

[5 marks]

(h) Define a binary predicate `maxHead(+List1,?List2)` that given a list `List1` of numbers returns a list `List2` with the same members and length, but with the head of `List2` being the maximum of all numbers in `List2`. For example,

```
?- maxHead([1,2,0,3],L).
L=[3,0,2,1]
```

[15 marks]

3. (a) Define a Definite Clause Grammar (DCG) for the set of strings $0^n1^m2^{n+m}$ of length $2n+2m$ for $n, m \geq 0$, using extra arguments to keep track of the numbers of 0's, 1's and 2's. For example, the DCG should accept 011222 but not 012.

[10 marks]

(b) Modify your DCG in part (a) so that it does *not* use extra arguments.

[10 marks]

(c) What are difference lists and how are they useful?

[7 marks]

(d) Making the difference lists explicit, write in ordinary Prolog notation the DCG rule

```
s --> [0],s,[2].
```

[8 marks]

(e) Define a DCG for the set of strings of the form www^{rev} where w is a string over the alphabet $\{0, 1\}$, and w^{rev} is w in reverse. For example, the DCG should accept 101001 (as $w = 10$, $w^{rev} = 01$) but not 100101.

[15 marks]