York University- Department of Computer Science and Engineering

SC/CSE 3401 3.00 – Functional and Logic Programming

Solutions to assignment 2

---

1) (12 marks) Consider a sequence of numbers, S= 1, 1, 3, 5, 11, ... in which $a_n=a_{n-1} + 2 * a_{n-2}$ and $a_0=1$, $a_1=1$.

(a) (5 marks) Use accumulators to write predicate *seq(a,b)* which is true for two consecutive numbers in this series. For example:

```
?- seq(1,1).
true
?- seq(3, X).
X=5
?- seq(X,Y).
X=1; Y=1;
X=1; Y=3;
X=3; Y=5; …
```

Answer.

```
seq(X,Y) :- seq(1, 1, X, Y).
seq(X, Y, X, Y).
seq(A, B, X, Y) :- next(A, B, C), seq(B, C, X, Y).
next(A, B, C) :- C is 2 * A + B.
```

(b) (4 marks) When will backtracking be problematic in your code? Give an example. How can adding cut to above code help to resolve this problem? How will the above example queries be answered after adding cut, and why?

Answer.

Backtracking is problematic in all queries except of form :- seq(X, Y). Beyond the first answer, the code will keep generating the sequence without ever reaching a stopping condition. For example, if we enter ; after the first solution to the query:

:-seq(3,X).

X=5;

?????

Prolog will not be able to stop.

Changing the second line of code to seq(X, Y, X, Y) :- !. will solve this problem since Prolog will not look for multiple solutions. The side effect is that there will be only one solution to queries such as
:- seq(X, Y). or even :- seq(1,X).

(c) (3 marks) Correct your code in a way to resolve the backtracking problem and still be able to get answers to above example queries. Hint: Use the fact that the numbers are increasing in this sequence and the built-in predicate nonvar. Try help(nonvar) to see Prolog's help on this predicate.

Answer.
Here is a way to solve the backtracking problem without compromising multiple solutions:

```
seq (X,Y) :- seq (1, 1, X, Y).
seq(A, _, X, _) :- nonvar(X), A > X, !, fail.
seq(_, B, _, Y) :- nonvar(Y), B > Y, !, fail.
seq(X, Y, X, Y).
seq(A, B, X, Y) :- next(A, B, C), seq(B, C, X, Y).
next(A, B, C) :- C is 2 * A + B.
```

---

2) (8 marks) In this question, you will be writing a coder/decoder for files.
(a) (3 marks) *Write encode(X, Y, Key)* that encodes a character X to a character Y using Key. This is done by adding Key (a small integer input) to the ASCII code of X to get the ASCII code of Y. For example:

```
:- encode('A', Y, 2).
Y= 'C'.
```

Hint: Use built-in predicate char_code.

Answer.

```
encode(X, Y, Key) :-  char_code(X, Xcode),
                      Ycode is Xcode + Key,
                      char_code(Y, Ycode).
```

(b) (2 marks) Write code that converts input by user to its coded version given a key. For example:

```
?- mystify(5).
assignment
fxxnlsrjsy
```

Write the code as a loop ending when end_of_file is read. (Exit by ctrl+D when you are testing the code.)

Answer.

```
mystify(Key) :- get_char(X), get_more(X, Key).
get_more(end_of_file, _):- !.
get_more(X, Key):-  encode(X,Y, Key), put_char(Y),
                    get_char(X2), get_more(X2, Key).
```

(c) (3 marks) Write *codefile(File1, File2, Key)* to code/decode a file. This code will read File1 character by character, writing encoded characters in File 2.

For example, if we have in file asg2_ sample.txt:
I am testing Assignment 2.

The query **?- codefile('asg2_ sample.txt', 'asg2_ sample.cod', 5).** will create file asg2_ sample.cod with the following text in it:
N%fr%yjxynsl%Fxxnlsrjsy%73

Note that the query **?- codefile('asg2_sample.cod', 'asg2_ sample.dcd', -5).** will decode the file.

Answer.

```
codefile(SrcFile, DstFile, Key) :-
                    open(SrcFile, read, X),
                    open(DstFile, write, Y),
                    current_input(SI), current_output(SO),
                    set_input(X), set_output(Y),
                    mystify(Key),
                    close(X), close(Y),
                    set_input(SI), set_output(SO).
```

---

3) (10 marks)Write *FindN(X, L, N)* to count how many X elements are in L and return as N, including element of elements. Assume X is a number and L is a list.

For example, the following query asks for the number of 3's in list L=[1,[2, 3, [4,3]],3, [3], []].

```
?- findN(3,  [1, [2, 3, [4,3]],3, [3], []],   N).
N = 4.
```

Answer.

```
findN(X, L, N) :- findN(X, L, 0, N).
findN(_, [], A, A).
findN(X, [H|T], A, N) :- findN(X, H, A, A1), findN(X, T, A1,
N),!.
findN(X, X, A, N):- N is A+1.
findN(X, Y, A, A) :- \+(X=Y).
```

---

4) (10 marks) Write *deepList(L1, L2)* which given list L1,returns list L2 with the same structure as L1, replacing each element with a number showing how deep in the list that element is. Use difference lists to get L2.

Examples:

```
?- deepList([a, [b]], L).
L = [0, [1]].
?- deepList([a, [b, c, []]], L).
L = [0, [1, 1, 1]].
?- deepList([a, [b, c, [d, e]], f], L).
L = [0, [1, 1, [2, 2]], 0].
```

Answer.

```
deepList(L1, L2) :- deepList(L1, 0, [], L2).

deepList([], _, Hole, Hole).
deepList([H|T], I, Hole, L) :-  deepItem(H, I, Hole1, L),
                                deepList(T, I, Hole, Hole1).

deepItem([H|T], I, Hole, [L|Hole]) :- !, J is I+1,
                                deepList([H|T], J, [], L).
deepItem(X, I, Hole, [I|Hole]).
```

---

5) (15 marks) In this question, you will be writing a very simple board game.

(a) (3 marks) Write *showBoard(B)* which shows a board on the screen. B is a board of 9 positions, shown as b(B1, B2, ..., B9), similar to our tic-tac-toe example in class.

Answer.

(a) Here is a very simple way to show the board.

```
showBoard(b(B1,B2,B3, B4,B5,B6, B7,B8,B9)) :-
            write(B1), write(' | '),
            write(B2), write(' | '),
            write(B3), nl,
            write(B4), write(' | '),
            write(B5), write(' | '),
            write(B6), nl,
            write(B7), write(' | '),
            write(B8), write(' | '),
            write(B9), nl.
```

(b) (1 mark) Write *b2List(B,L)* which can convert a board to a list and vice versa.

Answer.

Without using the built-in predicates:

```
b2list( b(B1,B2,B3, B4,B5,B6, B7,B8,B9) , [B1,B2,B3, B4,B5,B6,
B7,B8,B9] ).
```

If we could use the built-in predicates, we would simply write:

```
B2list(Board, L):- Board =..[_|L].
```

(c) (2 marks) Write *randomL(L,N)* which generates a list of length N with elements being 0 or 1 randomly. Use built-in predicate random.

Answer.

```
randomL([], 0).
randomL([H|T],N) :- H is random(2), N1 is N-1, randomL(T, N1).
```

(d) (2 marks) Write *generate(B)* that can generate a random board of 0's and 1's.

Answer.

```
generate(B) :- randomL(L, 9), b2list(B,L), !.
```

(e) (5 marks) Write *test(B)* that tests a board for a line of 1's or a line of 0's and announces the winner.

```
line(b(X,Y,Z, _,_,_, _,_,_), X, Y, Z).
line(b(_,_,_, X,Y,Z, _,_,_), X, Y, Z).
line(b(_,_,_, _,_,_, X,Y,Z), X, Y, Z).
line(b(X,_,_, Y,_,_, Z,_,_), X, Y, Z).
line(b(_,X,_, _,Y,_, _,Z,_), X, Y, Z).
line(b(_,_,X, _,_,Y, _,_,Z), X, Y, Z).
line(b(X,_,_, _,Y,_, _,_,Z), X, Y, Z).
line(b(_,_,X, _,Y,_, Z,_,_), X, Y, Z).

win1(1,1,1).
win0(0,0,0).

test(B) :- line(B, X,Y,Z), win1(X,Y,Z), write('1 wins'), nl, !.
test(B) :- line(B, X,Y,Z), win0(X,Y,Z), write('0 wins'), nl, !.
test(_) :- write('No winners!'), nl.
```

(f) (2 marks) Using generate and test paradigm, write code for randomly generating a board and announcing the winner.

Examples:

```
?- go.
1 | 0 | 0
1 | 1 | 1
0 | 0 | 0
1 wins
true.
```

Answer.

```
go :- generate(B), showBoard(B), test(B).
```