Department of Mathematics and Computer Science.

CSU 2280 – Deductive Reasoning and PROLOG for Artificial Intelligence

Practical Guide

Please use this guide to do practical in the regional elementary computer Laboratory.

Author: B. Hettige

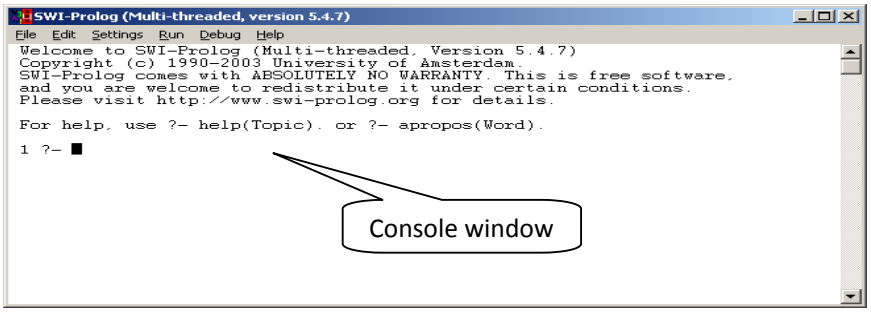
1. Basic in Prolog

1.1 Introduction to SWI PROLOG

Swi-Prolog can be freely downloaded from http://www.swi-prolog.org and can easily install it.

1.2 Run SWI-PROLOG

Click: start -> Program files -> SWI-PROLOG Prolog

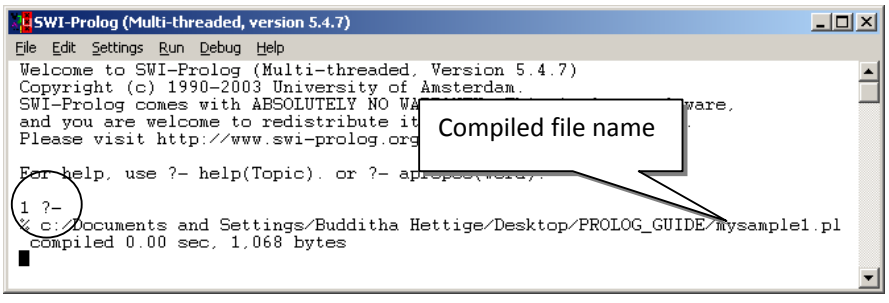


1.3 Create a source file

1. Open Text pad Using Windows start menu
2. Enter some text in your text document
3. Save your source file as a pl extension (mysample1.pl)

1.4 Consult a source file

1. In a prolog window click file menu
2. Click consult
3. Select appropriate file and press open button



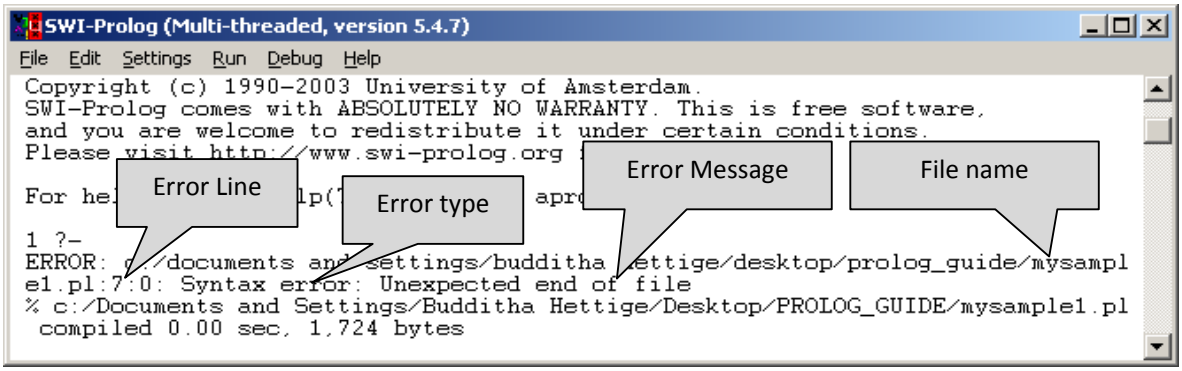
1.5 Update your source file and Re-consult

1. File - consult -select mysample1.pl file - open
2. Type consult(test.pl). in the prompt

1.6 Start SWI-PROLOG help

Help - online manual

1.7 Understanding the compilation Errors



*Note that if there is an error on your source program prolog does not compile rest

1.8 Create a Simple facts in a knowladge base

The Knowledge Base is simply a collection of facts. Facts are used to state things that are unconditionally true of the domain of interest. For example, we can state that Saman, ruwan and kamal are students, and kamal play cricket, using the following

```
student(saman).
woman(ruwan).
woman(kamal).
playcricket(kamal).
```

In general a fact has a name and values. The general format is

```
fact_name (data1, data2, … ).
Student('Saman Kumara').
Student('Saman kumara', '34/1, Colombo 3', 'as1003').
course('csu2280', 'Deductive Reasoning and PROLOG  for AI').
```

Create a Prolog Facts to store student name, age and sex.

```
student(saman,12,male).
```

1.9 Create rules

Rules state information that is conditionally true of the domain of interest. For example,

```
print :- write ('welcome to SWI-PROLOG').
printName(ID ) :- Student(Name, _,ID), write( Name).
```

Format of Prolog Rules format:

```
ruleName(Arguments..)   :- rule1,rule2.
```

```
? – print. ↵
        welcome to SWI-PROLOG'
        yes.

?- printName(as1003). ↵
        Saman kumara
        Yes.
```

?- printName(as1001). ↵
No.

2. List Operations

2.1 Create a Simple list

[a,b,c,d,e]
[saman, ruwan, kamal]

2.2 Print list

```
printList([]).
printList([H|T]) :- write(H),nl,printList(T).
```

2.3 Print list in reverse order

```
printList([]).
printList([H|T]) :- printList(T), write(H),nl.
```

2.4. Member in a list

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

2.5 Print length of a list

```
Length([],0).
length([H|T],S) :- length (T,S1),S is 1 + S1.
```

2.6 Print summation of the number list

```
sum([],0).
sum([H|T],S) :- sum(T,S1),S is H + S1.
```

2.7 Append two lists

```
appendLst([],L,L).
appendLst ([X|L1],L2, [X|L3]) :- appendLst (L1,L2,L3).
```

2.7 Delete an item in a list

```
delItem(X,[X|T],T).
delItem(X, [ Y|T], [Y|T]) :- delItem(X,T,T).
```

Some useful SWI-PROLOG predicates

1. **Dynamic** : Informs the interpreter that the definition of the predicate(s) may change during execution (:- dynamic student/1, course/2.)
2. **Multifile** : Informs the system that the specified predicate(s) may be defined over more than one file. (:- multifile student/1, course/2.)
3. **ensure_loaded** : If the file is not already loaded, this is equivalent to consult/1. Otherwise, if the file defines a module, import all public predicates. ensure_loaded('test.pl').
4. **Shell** : Execute Command on the operating system. (shell('notepad').)
5. **string_concat** : concatenate two string (string_concat(S1,S2,S).)

3. PROLOG Data bases

3.1 Create a Simple database to Store following information

Student:- store name, age , sex, address, email address and student ID
student('Saman kumara, 26, male, 'colombo 3', 'saman@yahoo.com', 'stu2201').
course :- course code name and unit
course(csu2280, 'Deductive Reasoning and PROLOG for AI', 30).
examResult :- coursecode, Student ID, Marks
examResult(csu2280, stu2201, 56).

3.2 Add new Recode – using assert/1

The following example shows a sample rule to add a new Student

```
AddstuResult :-
            write('Enter Course Code'),read(C),
            write('Enter Student ID'),read(ID),
            write('Enter Marks '),read(M),
            assert(examResult(C,ID,M)).
```

Create rules to add a new Student and a new course

3.3 Delete existing Records

The following example shows a sample rule to delete exam result;

```
deleteResult :-
            write('Enter Course Code'),read(C),
            write('Enter Student ID'),read(ID),
            retract(examResult(C,ID,_)).
```

3.4 Update records

The following example shows sample rule for update existing exam result

```
updateResult :-
            write('Enter Course Code'),read(C),
            write('Enter Student ID'),read(ID),
            examResult(C,ID,OM),
            retract(examResult(C,ID,OM)),
            write('Enter Marks '),read(NM),
            assert(examResult(C,ID,NM)).
```

3.5 Select Records

Select Student ID list how has followed csu2280 course

```
printList([]).
printList([H|T]) :- write(H),nl,printList(T).

stuList(Cou) :-   Setof(ID, ^Mark examResult(Cou,ID,Mark), List),
                  printLst(List).

stuList(Cou) :-   bagof(ID, ^Mark examResult(Cou,ID,Mark), List),
                  printLst(List).
```

3.6 To Checks whether a student is already in the database

```
checktudent  :-        write('Enter Name '), read(N),
                       (  student(N,_,_,_,_,_)
                          ->
                               write('Student Found')
                               ;
                                    write(' NO Student')
                               ).
```

4. File Handling

4.1 Create a text file to write a list of Student names

```
writetList([], Key).
wrireList([H|T], Key) :- write(Key, H), nl, writeList(T, Key).

writestuList(Cou , Key) :-      Setof(ID, ^Mark examResult(Cou,ID,Mark), List),
                                writeLst(List, Key).

writeStudentList :-             open('c:/studentList.txt', write, Key),
                                writestuList(Cou , Key)
                                close(Key).
```

4.2 Create a text file to append a list of Student names

```
writeStudentList :-             open('c:/studentList.txt', append, Key),
                                writestuList(Cou , Key)
                                close(Key).
```

4.3 read data from a text file

Create a simple txt file and save as sample.txt. in your text file, put 'end_of_file' in the last row.

Example:

```
saman
csu2280
23
end_of_file
```

Sample source to read a text file

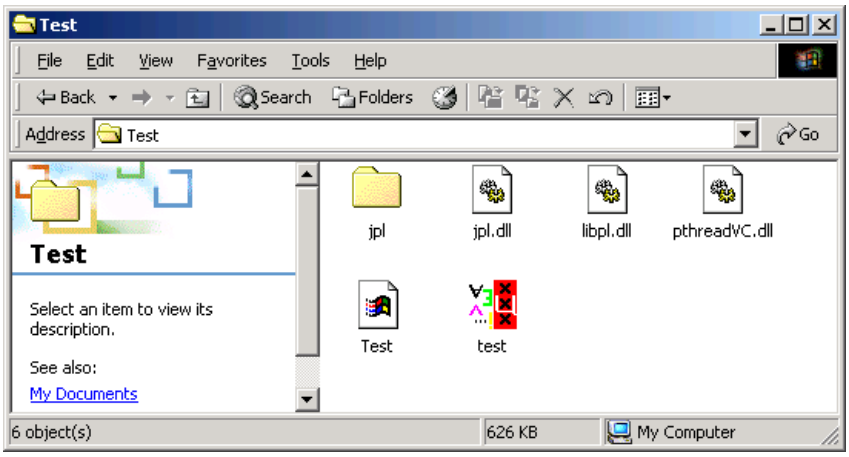
```
readWord(InStream,W) :-         get0(InStream,Char),
checkCharAndReadRest(Char,Chars,InStream),
                                atom_chars(W,Chars).
checkCharAndReadRest(10,[],_) :- !. % Return
checkCharAndReadRest(32,[],_) :- !. % Space
checkCharAndReadRest(-1,[],_) :- !. % End of Stream
checkCharAndReadRest(end_of_file,[],_) :- !.

checkCharAndReadRest(Char,[Char|Chars],InStream) :-  get0(InStream,NextChar),
                                checkCharAndReadRest(NextChar,Chars,InStream).
writeWord(end_of_file).
writeWord(X) :-                 write(X),nl.

readFile:-              open('c:/sample.txt', read, In), repeat,  readWord(In,W),  writeWord(W),
                        W == end_of_file,   !,  close(In).
```

Type the rule named readFile and press enter. The out put is shown bellow

```
?- readFile.
saman
csu2280
23
Yes
```

5. Java interfaces for SWI -PROLOG

To Create user interface by using java first you need to copy following files jpl.dll, libpl.dll, pthreadVC.dll and folder named 'JPl' these files are available in your SWI-Prolog folder.



Sample prolog source file 'test.pl'

```
student('as101', 'Budditha').
student('as102', 'Saman').

course(csu2280, as101, 76).
course(csu2280, as102, 56).
course(csu2280, as103, 45).
course(csu2279, as101, 78).
course(csu2279, as102, 29).

printList([]).
printList([H|T]) :- write(H),nl,printList(T).

stuList(Cou) :-     write('stu listRule Starting...'),nl,
                    setof(ID, Mark^ course(Cou,ID,Mark), List),
                    printList(List).

getList(Cou, List) :-   write('Get list Rule Start'),nl,
                    setof(ID, Mark^ course(Cou,ID,Mark), List).
```

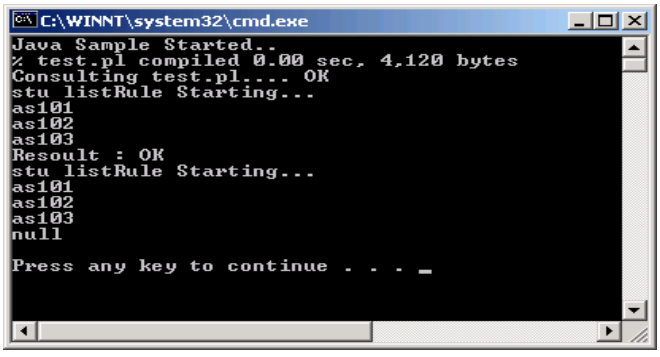
Java Sample program

```
/*
Test Sample for Java Prolog Interfaces
// Samaple java program
//********************
import java.util.Hashtable;
import jpl.Query;
import jpl.*;

public class Test
{
        Query q1;  // prolog output query
        Query q2;  // prolog output query
        Query q3;  // prolog output query
        public static void main( java.lang.String argv[] )
```

```
{
        run_testSample();
}

static void run_testSample()
{
 try
        {
          String name, outputStr;
          System.out.println("Java Sample Started.. ");
          Query q1 = new Query("consult('test.pl')."); // consult a file
          System.out.println("Consulting test.pl.... " + (q1.hasSolution() ? "OK" : "Failed"));

          Query q2 = new Query("stuList(csu2280)."); // run a fact
          System.out.println("Resoult : " + (q2.hasSolution() ? "OK" : "Failed"));
          Query q3 = new Query("getList(csu2280, Out)"); // run a rule

          java.util.Hashtable s2 = q2.oneSolution();
           outputStr = ""+ s2.get("Out");              // get solution
           outputStr= outputStr.replace('.',' ');
           outputStr= outputStr.replace('(',' ');
           outputStr= outputStr.replace(')',' ');
           outputStr= outputStr.replace('[',' ');
           outputStr= outputStr.replace(']',' ');
           outputStr= outputStr.replace('\'',' ');
           outputStr= outputStr.replace(',',' ');
            outputStr= outputStr.replace('`','.');
            outputStr= outputStr.trim();
            System.out.println(outputStr +"\n");
          }
        catch (Exception e)
        {
                System.out.println("ERROR" );
        } }}
```

Then compile test.java file and run it

The output of the java sample program is as follows



6. Some Prolog Sample programs

1. General examples
2. Net pay calculation program
3. Mathematical calculations
4. 2D Programming

6.1 Write the PROLOG Program for following task

i. Write a PROLOG program to convert Celsius temperature in to Fahrenheit
   F = (32 + C)9/5

ii. Write a PROLOG Program to find the Area and Volume of a cube.

iii. Find the Roots of the given equation { $ax^2 + bx + c = 0$}

iv. Create a Prolog Facts to store following information
   a. Student Name, index no, address
   b. IndexNo, course Code, Marks

v. Write Prolog predicates for above two Facts
   a. addStudent
   b. addresult

vi. Print grade for a given mark
   [M >= 70 A, 65 >= M < 70 B, 55 >= M < 65 C, 40 >= M < 55 S, M < 40 F]

vii. Find average mark of a given student.

viii. Print Result sheet for a given Student in the following format

```
Student Name: Sama
----------------------
No      Subject Grade
----------------------
1.      CSU1180       B
…       ……            …
Average               C
```

ix. Write a Prolog procedure starmap(N) to print N number of stars on the screen.
   For example,
   ? Starmap(7)
   ****
   Where N=7

x. Extend the above program to draw a bar chart. Your output must appear as follows. Then explain how it works.

```
barchart([1,9,3,5]).
1 *
9 *********
3 ***
5 *****
```

6.2 Net pay calculation program

ABC Bank stores customers' record by using two Prolog clauses. These Prolog clauses are given bellow.

:- dynamic customer/2, Account/3.
customer('Kumara', 'Panadura' sa01')
customer('Perera', 'Colombo', sa02')
customer('Kumara', 'Nugegoda', sa03")
account(sa01, c, 75000)
account(sa02, c,280000)
account(sa01, s,125000)

a. Write a Prolog rule named *'addNewCus/0'* to add new Customer. Hint: use *read/1* Prolog predicate to read data from keyboard.
b. Write a Prolog rule named *'addAccount/0'* to add new Customer account.
c. Write a Prolog rule named *'updateAcout/0'* to update Customer account (you can change the *account balance)*
d. Write a Prolog rule named *'printCus/0'* to print the customer name in ascending order. Your output should be as follows.

```
Customer Nanme
--------------
Jayalath
Perera
```

e. Write a Prolog rule to calculate interest for a given account. Calculation procedure is as follows:

Serving Account

BALANCE < 100,000 interest 5 %
BALANCE < 1000,000 and BALANCE >= 100,000 interest 7 %
BLANCE > 1000,000 interest 8 %

Current Account

BALANCE < 100,0000 interest 7 %
BALANCE < 1000,000 and BALANCE > 100,000 interest 8 %
BLANCE > 1000,000 interest 9 %

f. Write a Prolog rule named *'InterestSheet/1'* to print Interest of a given customer. Your output should be the following format.

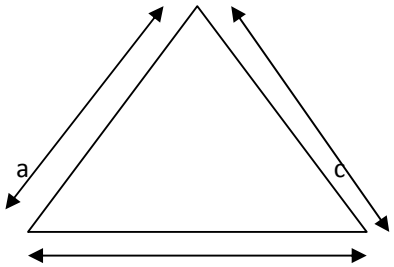
```
          INTERSEST SHEET
CUSTOMER ID                SA01
-------------------------------------------------
Acoun type                 Interest
-----------------------------
S                          xxx
C                          xxx
-----------------------------
```

Sample Program

```
:- dynamic customer/2, account/3.

customer('Kumara','Panadura', 'SA01').
customer('Perera','Colombo', 'SA02').
customer('Kumara', 'Nugegoda','SA03').

account('SA01', 'C', 75000).
account('SA02', 'C', 280000).
account('SA01', 'S', 125000).
```

```
addNewCus:-
      write('Enter Customer Name'),read(N),
      write('Enter Address '),read(A),
      write('Enter Cus ID '),read(I),
      assert(customer(N,A,I)).

addAcount:-
      write('Enter Customer ID '),read(I),
      write('Enter Account Type'),read(T),
      write('Enter Amount '),read(A),
      assert(account(I,T,A)).

updateAcount :-
            write('Enter Custom ID '), read(I),
            write('Enter Acount Type '), read(T),
            write('Add amount '), read(A),
            account(I,T,OA),
            retract(item(I,T,OA)),
            NA is A + OA,
            assert(account(I,T,NA)).

printList([]).
printList([H|T]) :- write(H),nl,printList(T).

printCus :- write('Customer Name'),nl,
             write('------------'),nl,
             setof(N,A^ I^ student(N,A,I),L),printList(L).

calInt(I,T,INT) :-      account(I,T,A),
                        calculateint(A,T,INT).

calculateint(A,'C',INT)  :- A < 100000,INT is A * 0.05.
calculateint(A,'C',INT)  :- A >=100000,A < 1000000, INT is A * 0.07.
calculateint(A,'C',INT)  :- A >= 1000000,INT is A * 0.08.

calculateint(A,'S',INT)  :- A < 100000,INT is A * 0.07.
calculateint(A,'S',INT)  :- A >=100000,A < 1000000, INT is A * 0.08.
calculateint(A,'S',INT)  :- A >= 1000000,INT is A * 0.09.

printLs(_,[]).
printLs(I,[H|T]) :-             account(I,H,_),
                        calInt(I,H,INT),
                        write(H),write('      '),write(INT),nl,
printLs(I,T).

interestSheet(I):-      write('-----------------'),nl,
                  write('  INTERSEST SHEET'),nl,
                  write('-----------------'),nl,
                  write('Customer ID : '),write(I),nl,
                  write('-----------------'),nl,
                  write('Type     Interest'),nl,
                  write('-----------------'),nl,
                  setof(C,A^ account(I,C,A),L),printLs(I,L),nl.
```

6.3. Mathematical calculations



Write a prolog predicate named *read_values/3* to read three length a,b and c from keyboard and Calculate perimeter of the triangle
Write a prolog rule to identify following triangle types

i. Equilateral triangle
ii. Isosceles triangle
iii. Unequal triangle

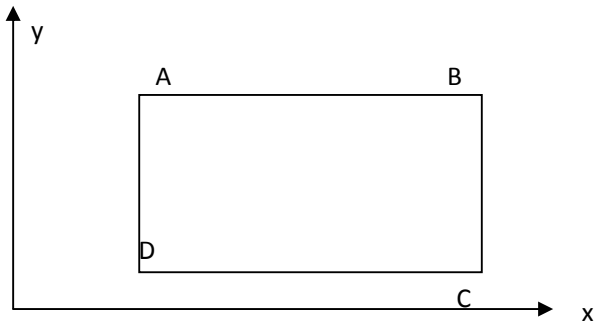
Create a prolog predicate to identify right angled triangle

i. Find the maximum length
ii. Check if triangle is 'right angled triangle' Hint: use Pythagoras low

```
:-dynamic student/2, result/3.

read_values(A,B,C) :- write('Enter first  : '),read(A),
                      write('Enter second : '),read(B),
                      write('Enter third  : '),read(C).

cal_area(A,B,C) :-   P is A + B + C , write('primeter is
'),write(P).


trangle(A,A,A)  :-   write('Equilateral triangle').

trangle(A,A,_)  :-   write('Isosceles triangle').
trangle(A,_,A)  :-   write('Isosceles triangle').
trangle(_,A,A)  :-   write('Isosceles triangle').

trangle(_,_,_)  :-   write('Unequal triangle').


max_val(A,B,C)  :-   A >= B , A >= C , write('Max value'), write(A).
max_val(A,B,C)  :-   B >= A , B >= C , write('Max value'), write(B).
max_val(A,B,C)  :-   C >= B , C >= A , write('Max value'), write(C).


right_tra(A,B,C):-  V1 is A * A, V2 is B * B , V3 is C * C , V1 is
V2 + V3, write('right traingle').
right_tra(A,B,C):-  V1 is A * A, V2 is B * B , V3 is C * C , V2 is
V1 + V3, write('right traingle').
right_tra(A,B,C):-  V1 is A * A, V2 is B * B , V3 is C * C , V3 is
V1 + V2, write('right traingle').

right_tra(_,_,_):-  write('NOT right traingle').
```

6.4. Graphics

A,B,C,D are the vertices of a rectangle.



Each point of the rectangle can be represented by point(X, Y).

A = point(1.0,6.5).
B = point(6.5,6.5).
C = point(6.5,1.5).
D = point(1.0,1.0).

a. Write a prolog predicate to calculate the length and width of the rectangle.
linelen(point(X1,Y1), point(X1,Y2), Len)

b. Calculate the perimeter of the above object.
cal_perm(P1, P2, P3, P4).

c. Calculate the area of the above object
cal_area(P1, P2, P3, P4).

d. Write prolog rules to identify the following object types.
   i. rectangle (if length 1 ≠ Length 2)
   ii. square (if Length 1 = length 2)

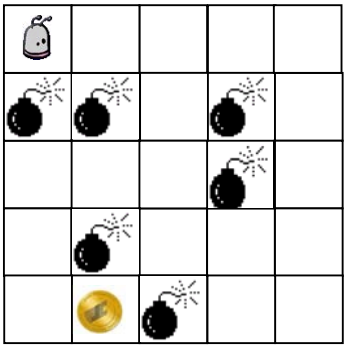
e. Calculate lengths AB and AD (in meters), and convert them to yards.
1 meter = 1.093yard

7. Artificial Intelligent Sample programs

1. Path finder
2. Farmer, Wolf, Goat and Cabbage Problem
3. 8-Quens problems
4. Tic-Tac-Toe in Prolog

7.1. Path finder

This Simple path finding program demonstrate how Prolog program can be used to Find Some given gols



Sample Program

```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).

printLst([]).
printLst([H|T])  :- printLst(T),write(H),nl.

go(Start,Goal)   :-    path(Start,Goal,Start).
path(Goal,Goal,L)     :-    write('Solution Path is: ' ), nl,
                            flatten(L,X),printLst(X).

path(State,Goal,L)    :-    move(State,Next),
                            not(member(Next,L)),
                            path(Next,Goal,[Next|L]),nl,!.

plsval(X,Y) :- Y is X + 1.
mulval(X,Y) :- Y is X - 1.

opp(X,Y) :- Y is X + 1.
opp(X,Y) :- Y is X - 1.

move(state(X,Y), state(X,Z)) :-
                            plsval(Y,Z),
                            not(notstate(X,Z)),
                            write('move to right'),nl.

move(state(X,Y), state(X,Z)) :-
                      mulval(Y,Z),
                            not(notstate(X,Z)),
                            write('move to left'),nl.

move(state(Y,X), state(Z,X)) :-
                            plsval(Y,Z),
                            not(notstate(Z,X)),
                            write('move to dowon'),nl.

move(state(Y,X), state(Z,X)) :-
                            mulval(Y,Z),
                            not(notstate(Z,X)),
```

```
                              write('move to up'),nl.

move(state(X,Y), state(X,Y)) :-
                 write('Backtrack'),nl, fail.

notstate(1,2).
notstate(2,2).
notstate(4,2).
notstate(4,3).
notstate(2,4).
notstate(3,5).

notstate(0,_).
notstate(_,0).
notstate(6,_).
notstate(_,6).
```

Example 1

```
?- go(state(1,1),state(2,5)).

      Solution Path is:
      state(1, 1)
      state(2, 1)
      state(3, 1)
      state(3, 2)
      state(3, 3)
      state(2, 3)
      state(1, 3)
      state(1, 4)
      state(1, 5)
      state(2, 5)
```

7.2. Farmer, Wolf, Goat and Cabbage Problem

A man is traveling with a goat, a wolf and a cabbage. They come to a river and the man wishes to get across the river with the goat, the wolf and the cabbage. There is a rowing boat on the bank of the river, but there is only room for one other thing on the boat as well as the man. If the wolf is left alone with the goat, he will eat it, and if the goat is left alone with the cabbage, he will eat it.

Sample Program

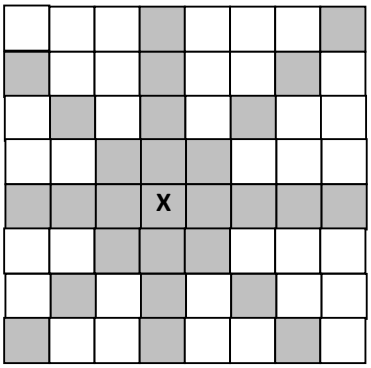
```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).

printLst([]).
printLst([H|T]) :- printLst(T),write(H),nl.

go(Start,Goal) :-     path(Start,Goal,Start).

path(Goal,Goal,L) :-  write('Solution Path is: ' ), nl,
                      flatten(L,X),printLst(X).

path(State,Goal,L) :-       move(State,Next), not(member(Next,L)),
                            path(Next,Goal,[Next|L]),nl,!.

opp(e,w).
opp(w,e).

move(state(X,X,G,C), state(Y,Y,G,C))
              :- opp(X,Y), not(unsafe(state(Y,Y,G,C))),
                 write('try farmer takes wolf'),nl.

move(state(X,W,X,C), state(Y,W,Y,C))
              :- opp(X,Y), not(unsafe(state(Y,W,Y,C))),
                 write('try farmer takes goat'),nl.

move(state(X,W,G,X), state(Y,W,G,Y))
              :- opp(X,Y), not(unsafe(state(Y,W,G,Y))),
                 write('try farmer takes cabbage'),nl.

move(state(X,W,G,C), state(Y,W,G,C))
              :- opp(X,Y), not(unsafe(state(Y,W,G,C))),
                 write('try farmer takes self'),nl.

move(state(F,W,G,C), state(F,W,G,C))
              :- write('BACKTRACK'),nl, fail.

unsafe(state(X,Y,Y,_)) :- opp(X,Y).
unsafe(state(X,_,Y,Y)) :- opp(X,Y).
```

Example

```
 go(state(w,w,w,w),state(e,e,e,e)).

 state(w, w, w, w)
 state(e, w, e, w)
 state(w, w, e, w)
 state(e, e, e, w)
 state(w, e, w, w)
 state(e, e, w, e)
 state(w, e, w, e)
 state(e, e, e, e)
```

6.3. 8 Queans problem

The challenge is to set 8 queens on a 8x8 grid so that no queen can "take" any other queen. Queens can move horizontally, vertically, or along a (45%) diagonal.

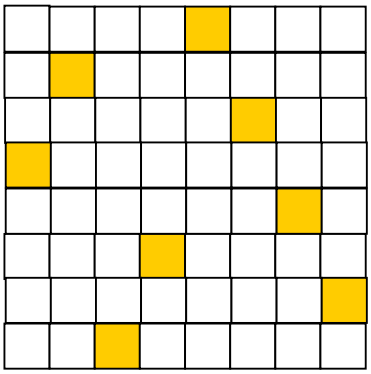


The following prolog program is developed to demonstrate where the queens can be placed in the 8 x 8 grid.

```
/* Chess queens challenge puzzle */

perm([X|Y],Z) :- perm(Y,W), takeout(X,Z,W).
perm([],[]).

takeout(X,[X|R],R).
takeout(X,[F|R],[F|S]) :- takeout(X,R,S).

solve(P) :-
     perm([1,2,3,4,5,6,7,8],P),
     combine([1,2,3,4,5,6,7,8],P,S,D),
     testfun(S),testfun(D).

combine([X1|X],[Y1|Y],[S1|S],[D1|D]) :-
     S1 is X1 +Y1,
     D1 is X1 - Y1,combine(X,Y,S,D).
combine([],[],[],[]).

testfun([X|Y]) :-  not(member(X,Y)), testfun(Y).
testfun([X]).
```

The program can be tested by using solve/1 predicate. A one solution is given bellow.

```
?- solve(P).

P = [5, 2, 6, 1, 7, 4, 8, 3]
```



6.4 Tic-Tac-Toe in Prolog

tick tack toe is a pencil-and-paper game for two players, O and X, who take turns marking the spaces in a 3×3 grid, usually X going first. The player who succeeds in placing three respective marks in a horizontal, vertical or diagonal row wins the game.

Sample code

```
/*
Tic-Tac-Toe in Prolog
*/

:- dynamic o/1.
:- dynamic x/1.

/* the various combinations of a successful horizontal, vertical or diagonal line */

ordered_line(1,2,3).
ordered_line(4,5,6).
ordered_line(7,8,9).
ordered_line(1,4,7).
ordered_line(2,5,8).
ordered_line(3,6,9).
ordered_line(1,5,9).
ordered_line(3,5,7).

/*
we use the line predicate to complete lines (cf. below),
so the elements of an ordered_line may be completed in any order,
i.e. as permutations of (A,B,C)
*/

line(A,B,C) :- ordered_line(A,B,C).
line(A,B,C) :- ordered_line(A,C,B).
line(A,B,C) :- ordered_line(B,A,C).
line(A,B,C) :- ordered_line(B,C,A).
line(A,B,C) :- ordered_line(C,A,B).
line(A,B,C) :- ordered_line(C,B,A).

/* a move to choose, i.e. field to occupy, should be good
  according to some strategy and the field should be empty  */

move(A) :- good(A), empty(A).

/* a field is empty if it is not occupied by either party */

full(A) :- x(A).
full(A) :- o(A).

empty(A) :- not(full(A)).

%%% strategy

good(A) :- win(A).
good(A) :- block_win(A).
good(A) :- split(A).
good(A) :- block_split(A).
good(A) :- build(A).

%%% default case of picking the center, the corners and the sides
%%% in that order
```

```
good(5).
good(1). good(3). good(7). good(9).
good(2). good(4). good(6). good(8).

%%% we win by completing a line

win(A) :- x(B), x(C), line(A,B,C).

%%% we block the opponent to win by completing his possible line

block_win(A) :- o(B), o(C), line(A,B,C).

%%% a split creates a situation where opponent can not block a win in the next move

split(A) :- x(B), x(C), different(B,C), line(A,B,D), line(A,C,E), empty(D), empty(E).

same(A,A).
different(A,B) :- not(same(A,B)).

%%% block opponent from creating a split

block_split(A) :- o(B), o(C), different(B,C), line(A,B,D), line(A,C,E), empty(D), empty(E).

%%% simply pick a square that builds towards a line

build(A) :- x(B), line(A,B,C), empty(C).

%%%*****************************************************************
%%% predicates to interactively run the game

%%% when is game definitely over?
all_full :- full(1),full(2),full(3),full(4),full(5),full(6),full(7),full(8),full(9).

%%% options for earlier success

done :- ordered_line(A,B,C), x(A), x(B), x(C), write('I won.'),nl.

done :- ordered_line(A,B,C), o(A), o(B), o(C), write('You won.'),nl.

done :- all_full, write('Draw.'), nl.

%%% interaction

getmove :- repeat, write('Please enter a move: '),read(X), between(1,9,X), empty(X), assert(o(X)).

%%% the computer's moves

makemove :- move(X),!, assert(x(X)).
makemove :- all_full.

%%% printing the board

printsquare(N) :- o(N), write(' o ').
printsquare(N) :- x(N), write(' x ').
printsquare(N) :- empty(N), write('   ').

printboard :- printsquare(1),printsquare(2),printsquare(3),nl,
              printsquare(4),printsquare(5),printsquare(6),nl,
              printsquare(7),printsquare(8),printsquare(9),nl.

%%% clearing the board
```

```
clear :- x(A), retract(x(A)), fail.
clear :- o(A), retract(o(A)), fail.

%%% main goal that drives everything:

play :- not(clear), repeat, getmove, makemove, printboard, done.
```

To run your sample program type ‘play’

```
? play.
Please enter a move: 1.
 o
   x

Please enter a move: 3.
 o  x  o
   x

Please enter a move:
```