# CI342

## *Workshop 1*

This workshop is intended to re-introduce Prolog and to quickly 'get up to speed' on the basics of the language. All students on CI342, who have studied CI213 (Intelligent Systems) have had one year of exposure to the language (including Flex), and therefore Prolog should not be 'alien' to you. The workshop is split into 2 parts:

**Part 1.** Basic representation and reasoning in Prolog. This uses a scenario that was used in the workshop programme on CI213, and is based on a crime-solving world. It is extended, of course, and there are exercises designed to extend your knowledge/ familiarisation.

**Part 2.** Content and exercises targeted to the V-World system you will encounter next, including list representation and processing.

The latest Student (time-limited) version of WIN-PROLOG will be available in the Aldrich Library for you to copy/ install on your own PC if you choose to.

## *1.0 getting started Experimenting with a simple program*

**Start WIN-PROLOG** in the usual way

**Create a new Edit Window** from the File/New Menu. Make sure that you save previous programs first

**Use the following program**, making sure that the syntax is exactly like this if you are typing it in. You can download the program from:

P:\COURSEWK\CI342\ Workshops\Workshop1\extended_crimes.pl

Alternatively you can access to code (as a text file) on Student Central

You might recognise this program. It is simple Prolog, with facts and rules. Read the program and satisfy yourself that you know what should happen when it is run.

```
/* a simple program in Prolog. A crime mystery.

Notice also this form of commenting out
code. It works for multiple lines */

possible_suspect(fred).                          % start of domain facts
possible_suspect(mary).
possible_suspect(jane).
possible_suspect(george).
owes_money_to(bill,mary).

crime(robbery, john, tuesday_night, park).
crime(assault, robin, thursday, pub).
crime(robbery, jim, wednesday, pub).

was_seen(fred, tuesday_night, park).
was_seen(jane, thursday, pub).
admitted_was_at(john, thursday, pub).
owes_money_to(jane,robin).

jealous_of(fred, john).                          % end of domain facts

prime_suspect(Person, Crime):-                   % start of domain rules (axioms)
        crime(Crime, Victim, Time, Place),
        possible_suspect(Person),
        was_at(Person, Time, Place),
        had_motive_against(Person, Victim).

prime_suspect(unknown, Crime).

had_motive_against(Person, Victim):-
        jealous_of(Person, Victim);
        owes_money_to(Person, Victim).

was_at(Person, Time, Place):-
        was_seen(Person, Time, Place).

was_at(Person, Time, Place):-
        admitted_was_at(Person, Time, Place).    % end of domain rules
```

**Exercises**

Input the following goals, using whichever debugger you prefer, and select Trace to explore the way Prolog deals with them. Try to follow Prolog's reasoning:

a) prime_suspect(fred, robbery).
b) prime_suspect(Person, robbery). % then make the system look for other solutions
c) prime_suspect(Person, assault).
d) Experiment with other aspects of crime detection (be imaginative). For example there might be situations in which alibis may be relevant.

## 2.0 Experimenting with representation & reasoning using lists

The following program is a simplified version of the world representation in V-World. This (and the extensions to it) can be found on the P-Drive and on StudentCentral).

### Simple_world.pl

```
/*
simple list-based representation
based on V-World

Each of the facts below is the same clause vmap/3
Argument
1.     (0,0) this is the X,Y values that specify the LEVEL
       i.e. 0,0 for the current level, 1,0 for one level EAST
       2,0 for 2 levels east
       -1,0 for one level WEST
       0,1 for one level SOUTH
       0,-1 for one level NORTH
       (these are the protocols in V-World)

2.     0-7 (in the simplified case below) represent the Y coordinates
       of the world

3.     [w,w,w,...] These are objects in the X direction in the world
       i.e. the object at X,Y= 3,6 at level (0,0) is a tree
       Note: the list positions are numbered 1-8 (not 0-7)
*/

vmap((0,0),0,[w,w,w,w,w,w,w,w]).
vmap((0,0),1,[w,o,o,w,o,o,o,w]).
vmap((0,0),2,[w,o,o,w,o,o,o,w]).
vmap((0,0),3,[w,o,o,w,o,tree,o,w]).
vmap((0,0),4,[w,o,hornet,o,o,o,o,w]).
vmap((0,0),5,[w,o,o,o,o,bird,o,w]).
vmap((0,0),6,[w,o,o,o,o,o,o,w]).
vmap((0,0),7,[w,w,w,w,w,w,w,w]).
```

This may not be the 'best' representation, but it provides a very simple and immediately obvious world description, i.e. you can actually 'see' this 2-D world at Level (0,0). It is surrounded by:

**w** = wall

it has plenty of:

**o** = open space

and a couple of other things too:

**tree**
**hornet**

We can access this world quite easily. Do the following:
Open and compile the program above & type the following into the Console Screen

## ?- vmap(Level,Y,X).

And note the output.
Press the ';' key (or left click anywhere in the Console Screen) to force Prolog to look for any other bindings and note the output

Now to find out where the tree is. We can use member/3

member/3 Agument 1 = the term to be found in the list. In this case it is 'tree'
member/3 Agument 2 = the list to be searched. We need to provide this list
member/3 Agument 3 = the term's position in the list (if found)

We first need to make a binding for the list. We already have a way of doing this (above):

vmap(Level,Y,X). X in this query is bound to the 3rd argument in vmap/3, which is the X coordinate(s). We could usefully rename this variable 'List':

**vmap(Level,Y,List).**

Now, with a binding for List, we can use the member/3 predicate to find our tree. We can also rely on Prolog's built-in backtracking to look at all occurrences of vmap/3 until a tree is found or the query fails. Do this:

## ?- vmap(Level,Y,List), member(tree,List,Position).

Note the output.
Press the ';' key (or left click anywhere in the Console Screen) to force Prolog to look for any other bindings and note the output.

## 2.1 Inspecting the world

V-World has all kinds of access functions. The following code snippet is taken from that program and is useful to find out what object exists at a given level, X & Y coordinates

```
/* ..................................................
Start of inspect object at X,Y
Level = (0,0)
X is the position in the list starting at 1
Y is the Y-coordinate (2nd arg to vmap/3
Object is the variable to be bound, i.e the object at
that location.
*/

inspect(Level,X,Y,Object):-
      vmap(Level,Y,List),          % make the binding
      K is X+1,                    % member/3 references from zero
      member(Object,List,K).       % what exists at that location?

% End of inspect object at X,Y
% ..................................................
```

Try this utility out on any location you like. Note how useful the member/3 clause is in this situation.

## 2.2 placing and replacing objects in the world

This is ***totally based on what happens in V-World***. Again it may not be the best way of doing it, but it IS the way it is done.

Using the concept of the world being represented as a series of asserted facts, the placement/ replacement algorithm works as follows

1. Find the list which is given by the level and Y coordinates
2. Retract it
3. Make the replacement at the given position in this list. It is a replacement since at the very least it will have the object = open
4. Assert the new list

The following segment of code can be used on the world as described above in the series of vmap/3 facts:

```
/* replace_obj/4 places (replaces) the Object at the given
Level and X (Pos)/ Y coordinate.

It first finds and retracts the X List at that Level &
Y coordinate, does the object replacement & then asserts
the new list (Newlist)
*/

replace_obj(Level, Pos, Y, Obj):-
      retract(vmap(Level, Y, List)),
      replace(Obj,Pos,List,Newlist),
      assert(vmap(Level, Y,Newlist)).
```

```
/* This is both the boundary condition and the part that
actually does the placement/ replacement

Pos is the position in the relevant X coord list. Note it
is numbered from zero.

When this reaches zero, the correct position in the list
is found. Only at that point does the rest of the clause
perform its actions.

first it makes a binding to Rest, i.e. those objects that
follow the required position up to the end of the list
Then it puts Obj at the head of it, i.e. replacing anything
already there.

This is the end of recursion, so as it unwinds, it puts
back those objects it removed as it recursed to the
given position.
*/

replace(Obj, 0,[_|Rest],[Obj|Rest]).

/* This part does the search through the list and sets
up the lists for after recursion. It will continually
reduce the value of Pos until = 0, at which point the
boundary condition is satisfied (see above).
*/

replace(Obj,Pos,[X|Y],[X|Z]):-
      Pos > 0,
      M is Pos - 1,
      replace(Obj,M,Y,Z).
```

**Exercises**

1. **Place a ball in the world position Y=5, X=3. Note the X positions in this program run from zero.**
2. **Test the outcome of your placement by examining the relevant list (use the queries explored above).**
3. **Use Trace to 'get a feel' for what is going on. This is pretty sophisticated recursive list processing, but you should be able to see that lists are being manipulated in a certain way**

## 2.3 Beginning to explore V-World

Open the first simple world in V-World (bird1.vw)

**File/Open**… The default file type is .pl, but V-World worlds are saved using the suffix .vw, therefore select **All Files (*.*)** and then

## Bird1.vw

Notice that it is a simple Prolog file with the same format as our smaller one. This on is only a little larger - 22/16

Now look for the access functions we have been working with:

1. Open **vworld.pl** in the V-World Folder and use:

   **Search/ Find** in the Prolog menu bar

   Search for **vinspect(** **…**You will need to 'find next' until the main clause is found

   Have a look at this code. It is only slightly more complicated by its manipulation of levels

2. In **vworld.pl** again, do a similar search for **vreplace(**

   Again, examine the 'real' code here.

Subsequent workshops will be based totally on the V-World System. Spend some time looking at the material on StudentCentral and in the V-World Folder on the P-Drive.