Artificial Intelligence Lab Manual
CSEN 3286

Name : Tusita Sarkar.
Class : CSE-B,3rd
Roll No. : 1751087

Index

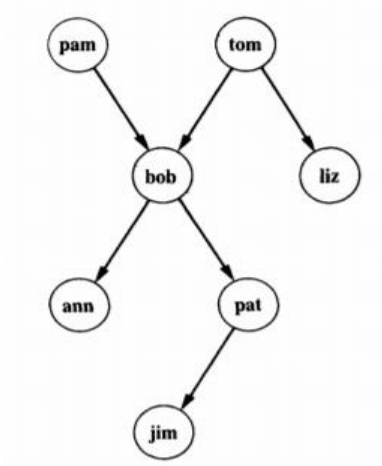
Sl.No.	Date	Title	Signature

DAY 1:

An example program: defining family relations

Prolog is a programming language for symbolic, non-numeric computation. It is especially well suited for solving problems that involve objects and relations between objects. Figure below, shows an example of a family relation. The fact that Tom is a parent of Bob can be written in Prolog as:

```
parent( tom, bob).
```



Here we choose parent as the name of a relation; tom and bob are its arguments.The whole family tree of the figure is defined by the following Prolog program:

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

This program consists of six clauses. Each of these clauses declares one fact about the parent relation.
For example, Is Bob a parent of Pat? This question can be communicated to the Prolog system by typing into the terminal:

?- parent(bob, pat).
true

Who is Liz's parent?
?- parent(X, liz).
X: tom

The grandparent rule would be, according to this convention, written as follows:

```
grandparent(X, Z) :- parent( X, Y), parent( Y, Z).
```

The predecessor relation, which consists of two rules: one for direct predecessors and one for indirect predecessors. Both rules are rewritten together here:

```
predecessor(X, Z) :-parent( X, Z).
predecessor(X, Z) :-parent( X, Y),predecessor(Y, Z)
```

Problem 1: A person Xmay steal Y if X is a thief, X is a man, X likes Y,Y is valuable. Given the following facts, define a predicate steal(X,Y) and determine who steals what. Facts: 1)John is a man. 2)Mary is woman. 3)Gold is valuable. 4)John likes Mary. 5)John likes gold. 6)John is a thief. 7)Mary is a thief.

Code:

```
man(john).woman(mary).
valuable(gold).
likes(john,gold). likes(john,gold).
thief(john). thief(mary).
steal(X,Y):-thief(X),man(X),valuable(Y),likes(X,Y).
```

Output:

?- steal(X,Y).
X = john,
Y = gold

Problem 2: Consider the following facts and rules: 1)Hardware is an easy course. 2) Books for hardware are available. 3) Logic is not an easy course 4) graphics is an easy course. 5) Graphics has 8 credits. 6) Graphics has lab components. 7) Book for data structure is available. 8) John takes compiler. X takes Y if Y is a easy course & books are available.
X takes Y if Y has 8 credits & and lab component.

1) Does John takes graphics course.
2) Which course does John takes?
3) Who takes graphics?

Code:

```
easy(hardware). easy(graphics). noteasy(logic).
booksavailable(hardware). booksavailable(data_structure).
credit8(graphics).student(john).
labcomp(graphics).
takes(john,compiler).
takes(X,Y):-
student(X),easy(Y),booksavailable(Y);student(X),credit8(Y),labcomp(Y).
```

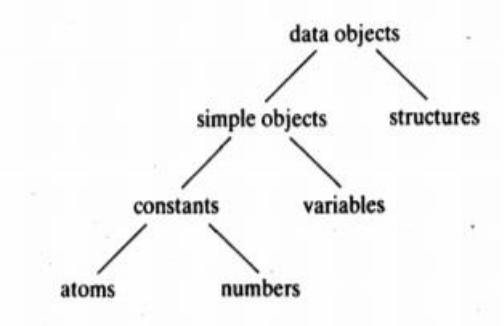
Output:

1)?- takes(john,graphics).
true.
2)?- takes(john,Y).
Y = compiler ;
Y = hardware ;
Y = graphics.
3)?- takes(X,graphics).
X = john.

DAY 2:

Data objects:

The Prolog system recognizes the type of an object in the program by its syntactic form. This is possible because the syntax of Prolog specifies different forms for each type of data objects.



Problem 1: Find the maximum of three numbers.

Code:

```
max(X,X,Y,Z):-X>=Y,X>=Z.
max(Y,X,Y,Z):-X=<Y,Y>=Z.
max(Z,X,Y,Z):-Z>=Y,X=<Z.
```

Output:

```
?-max(X,1,2,3).
X = 3.
?- max(X,1,4,3).
X = 4 .
?- max(X,6,4,3).
X = 6 .
?- max(X,2,2,2).
X = 2
```

Problem 2: Find the factorial of a number.

Code:

```
fact(0,1).
fact(N,R):-N>0,N1 is N-1,fact(N1,R1),R is N*R1.
```

Output:

```
?-  fact(6,R).
R = 720 .
```

Problem 3: Generate the Nth Fibonacci term.

Code:

```
fib(1,0). fib(2,1).
fib(N,R):-N>2,N1 is N-1,N2 is N-2,fib(N1,R1),fib(N2,R2),R is R1+R2.
```

Output:

?- fib(7,R).
R = 8

Problem 4: Find GCD of two numbers using Euclidean theorem.
Code:

```
gcd(A,B,Z):-A is 0,Z is B;B is 0,Z is A.
gcd(A,B,Z):-A>0,B>0,R is A mod B,gcd(B,R,Z).
```

Output:

?- gcd(2,3,R).
R = 1 .
?- gcd(4,12,R).
R = 4

Problem 5: Sum of N natural numbers.
Code:

```
sum(1,1).
sum(N,R):-N>1,N1 is N-1,sum(N1,R1),R is N+R1.
```

Output.

?- sum(6,R).
R = 21
?- sum(16,R).
R = 136

Problem 6: From the given graph. Find all the edges. Find those nodes having edges to node a. Write a Prolog program to check whether there is any path between two nodes.

Code:

```
edge(a,b).
edge(a,j).
edge(a,c).
edge(b,e).
edge(b,f).
```

```
edge(c,g).
edge(f,g).
edge(e,f).
edge(b,c).
path(X,Y):-edge(X,Y).
path(X,Y):-edge(X,Z),path(Z,Y).
```

Output:

?-edge(X,Y).
X = a,
Y = b
X = a,
Y = j
X = a,
Y = c
X = b,
Y = e
X = b,
Y = f
X = c,
Y = g
X = f,
Y = g
X = e,
Y = f
X = b,
Y = c
?- edge(X,a).
false
?- path(a,f).
true
true
false

Problem 7: Write a program to keep on asking random numbers from the user and it will keep on generating its cube.It stops taking number when the user wants to stop.

Code:

```
cube:-read(X),X\='STOP',R is X*X*X,write(cube:R),cube.
```

Output:

?-cube.
| 2.
cube:8
| 5.
cube:125
| STOP
false

Problem 8: Write a prolog program to design a login module while asks user for login name and his password. If the password is not correct it keeps on asking till the correct password is entered. Same for the login name.

Code:

```
info(ravi,132).
info(fatim,243).
info(dhruv,354).
info(hari,465).
info(rita,576).
info(riya,687).
log(X):-write('password:'),read(Y),not(info(X,Y)),
        write('invalid password!!'),log(X);write('login successful!!').
login:-write('login name:'),read(X),not(info(X,_)),
       write('invalid user!!'),login;log(X).
```

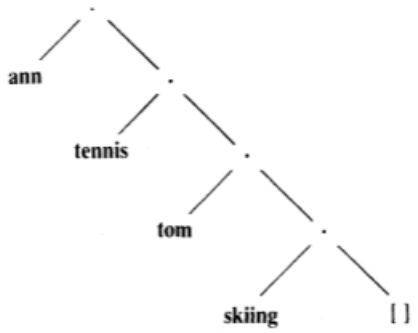
Output.

```
?- login.
login name:rey.
invalid user!!login name:|: rasi.
invalid user!!login name:|: ravi.
password:|: 454.
invalid password!!password:|: 754.
invalid password!!password:|: 132.
login successful!!
true .
```

DAY 3

The **List** is a simple data structure widely used in non-numeric programming. A list is a sequence of any number of items, such as ann, tennis, tom, skiing. Such a list can be written in Prolog as:

[ann, tennis, tom, skiing]



Tree representation of the list **[ann, tennis, tom, skiing]**.

To express this in the square bracket notation for lists, Prolog provides another notational extension, the vertical bar, which separates the head and the tail:

[Head | Tail]

or

[Iteml, Item2, ... | Others]

or

[a,b,c] = [a | [b,c]] : [a,b | [c]] =[a,b,c | []]

Problem1: Program to check whether an element is a member of the list or not.

Code:

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

Output:

```
?- member(4,[3,4,7,1]).
true .
?- member(9,[3,4,7,1]).
false.
```

Problem2: Program to find the length of list.

Code:

```
len([],0).
len([_|Y],N):-len(Y,N1),N is N1+1.
```

Output:

```
?- len([1,4,7,2,1,8],X).
X = 6.
?- len([],X).
X = 0.
```

Problem3: Find the maximum element of the list.
Code:

```
max(X,Y,X):-X>Y.
max(X,Y,Y):-X=<Y.
maxlist([],0).
maxlist([X],X).
maxlist([X|Y],M):-maxlist(Y,M1),max(X,M1,M).
```

Output:

?- maxlist([-8,-1,10,-2,-3,-4],M).
M = 10 .
?- maxlist([-1,-2,-3,-4],M).
M = -1

Problem4: Concatenate two lists.
Code:

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

Output:

?- conc([1,2,3,4],[5,7,6],L).
L = [1, 2, 3, 4, 5, 7, 6].
?- conc([],[5,7,6],L).
L = [5, 7, 6].
?- conc([4,5],[],L).
L = [4, 5].

Problem5: Program to display the last and second last elements of the list.
Code:

```
lastd([X,Y],X,Y).
lastd([_|Tail],X,Y):-lastd(Tail,X,Y).
```

Output:
?- lastd([1,2,7,4,6,1,3],SecondLast,Last).
SecondLast = 1,
Last = 3

Problem6: Remove the duplicate element of the list

Code:

```
mymember(X,[X|_]).
mymember(X,[_|T]) :- mymember(X,T).
set([],[]).
set([H|T],[H|Y]) :-not(mymember(H,T)),set(T,Y).
set([H|T],Y) :- mymember(H,T),set(T,Y).
```

Output:

?- set([1,2,3,3,5,6,3,6,2],X).
X = [1, 5, 3, 6, 2] .
?- set([1,2,3],X).
X = [1, 2, 3] .
?- set([1],X).
X = [1] .
?- set([],X).
X = [].

Problem7: Check whether a list is ordered.
Code:

```
ordered( [] ) .
ordered( [_] ) .
ordered( [X,Y|Z] ) :- X =< Y , ordered( [Y|Z] ) .
```

Output:

?- ordered([3,4,5,6]).
true .

Problem8: Reverse a list.
Code:

```
rev(L,List2):-findrev(L,[],List2).
findrev([],List1,List1).
findrev([X|Tail],List1,List2):-findrev(Tail,[X|List1],List2).
```

Output:

?- rev([1,2,3,4,5],L).
L = [5, 4, 3, 2, 1].
?- rev([1,2,3,4,5,6,1,2,4,6],L).
L = [6, 4, 2, 1, 6, 5, 4, 3, 2|...].