United College of Engineering & Research, Prayagraj

Artificial Intelligence Lab File (RCS-752)

Lab Incharge : Mr. Ramjee Dixit

Heramb Mishra
1801010047
Computer Science, 7-A

INDEX PAGE

S. No	Name of Practical	Date of Completion	Remark	Sign of Instructor
1	Study of Prolog			
	a. Simple programs Using Facts and Rules (i) greater among two numbers (ii) Write predicates One converts centigrade temperatures to Fahrenheit, the other checks if the temperature is below freezing.	23/9/2021 11/10/2021		
	b. Recursive Programs (i) Factorial of given number (ii) Fibonacci Series (ii) Tower of Hanoi (iii) Path in Graph	 18/10/2021 25/10/2021 1/11/2021 8/11/2021		
	C. Using List and Set (i) Basic Set Operations, Union, Intersection, and set difference (ii) Path in Graph	22/11/2021 29/11/2021		
2.	Write a program to solve the Monkey Banana problem.			
3.	Write a program to solve the 4-Queen problem.			
4.	Write a program to solve traveling salesman problem			
5.	Write a program to solve water jug problem using LISP			

a. Simple programs Using Facts and Rules

(i) greater among two numbers

```
max(X,Y):
(
 X=Y ->
  write('both are equal')
 ;
 X>Y ->
  (
  Z is X,
  write(Z)
  )
  ;
  (
  Z is Y,
  write(Z)
  )
).
```

(ii) Write predicates One converts centigrade temperatures to Fahrenheit, the other checks if the temperature is below freezing.

Rules:

```
c_to_f(C,F) :-
F is C * 9 / 5 + 32.
freezing(F) :-
F =< 32.
```

Output:

Queries:

```
?- c_to_f(100,X).
X = 212
Yes ?- freezing(15) .
Yes ?- freezing(45).
 No
```

b. Recursive Programs

(i) Factorial of given number

```
factorial(0,1).

factorial(N,F) :-
   N>0,
   N1 is N-1,
   factorial(N1,F1),
   F is N * F1
```

ii). Fibonacci series

```
fib(0, 0).
fib(X, Y) :- X > 0, fib(X, Y, _).
fib(1, 1, 0).
fib(X, Y1, Y2) :-
X > 1,
X1 is X - 1,
fib(X1, Y2, Y3),
Y1 is Y2 + Y3.
```

Output:
Goal:
?-fib(10,X).
X=55

iii) Tower of hanoi

```
move(1,X,Y,_) :-
  write('Move top disk from '), write(X), write(' to '), write(Y), nl.
move(N,X,Y,Z) :-
  N>1,
  M is N-1,
  move(M,X,Z,Y),
  move(1,X,Y,_),
  move(M,Z,Y,X).
```

iv) Path in graph

```
edge(a, b).
edge(a, d).
edge(a, e).
edge( a, f ).
edge( a, k).
edge(b, c ).
edge(c, h ).
edge(d, c ).
edge(e, g ).
edge( f, g).
edge( h, i).
edge(i, j ).
edge(k, i ).
edge(g, j ).
 % base condition
 path(A,B,[A,B]):-edge(A,B).
 path(A,C,[A|Z]):-edge(A,B), path(B, C, Z).
```

C. Using List and Set

(i) Basic Set Operations, Union, Intersection, and set difference

UNION

```
union([],[],[]).
union([], Y, Y).
union([X|T],Y,Z):-member(X,Y),union(T,Y,Z).
union([X|T], Y, [X|T1]):-union(T,Y,T1).
```

INTERSECTION

```
intersection([],[],[]).
intersection([], Y, []).
intersection([X|T],Y,[X|T1]):-member(X,Y),intersection(T,Y,T1).
intersection([X|T], Y, Z):-intersection(T,Y,Z).
```

DIFFERENCE

```
s_difference([],[],[]).
s_difference([], Y, []).
s_difference([X|T],Y, Z):-member(X,Y),s_difference(T,Y,Z).
s_difference([X|T], Y, [X|T1]):-s_difference(T,Y,T1).
```

2. Write a program to solve the Monkey Banana problem.

```
in_room(bananas).
in_room(chair).
in_room(monkey).
clever(monkey).
can_climb(monkey, chair).
tall(chair).
can_move(monkey, chair, bananas).
can_reach(X, Y):-
clever(X),close(X, Y).
get_on(X,Y):- can_climb(X,Y).
under(Y,Z):-
in_room(X),in_room(Y),in_room(Z),can_climb(X,Y,Z).
close(X,Z):-get_on(X,Y),
under(Y,Z);
tall(Y).
```

3. Write a program to solve 4-Queen problem.

```
domains

queen = q(integer, integer)
queens = queen*
freelist = integer*
board = board(queens, freelist, freelist, freelist, freelist)
predicates
nondeterm placeN(integer, board, board)
nondeterm place_a_queen(integer, board, board)
nondeterm nqueens(integer)
nondeterm makelist(integer, freelist)
nondeterm findandremove(integer, freelist, freelist)
nextrow(integer, freelist, freelist)

clauses

nqueens(N):-
makelist(N,L),
Diagonal=N*2-1,
makelist(Diagonal,LL),
placeN(N,board([],L,L,LL,LL),Final),
write(Final).
placeN(_,board(D,[],[],D1,D2),board(D,[],[],D1,D2)):-!.

placeN(N,Board1,Result):-
place_a_queen(N,Board1,Board2),
```

```
placeN(N,Board2,Result).
place_a_queen(N,
board(Queens,Rows,Columns,Diag1,Diag2),
board([q(R,C)|Queens],NewR,NewC,NewD1,NewD2)):-
nextrow(R,Rows,NewR),
findandremove(C,Columns,NewC),
D1=N+C-R,findandremove(D1,Diag1,NewD1),
D2=R+C-1,findandremove(D2,Diag2,NewD2).
findandremove(X,[X|Rest],Rest).
findandremove(X,[Y|Rest],[Y|Tail]):-
findandremove(X,Rest,Tail).
makelist(1,[1]).
makelist(N,[N|Rest]) :-
N1=N-1,makelist(N1,Rest).
nextrow(Row,[Row|Rest],Rest).
```

4. Write a program to solve traveling salesman problem

```
predicates
nondeterm road(town,town,distance)
nondeterm route(town,town,distance)
clauses
road("tampa","houston",200).
road("gordon","tampa",300).
road("houston","gordon",100).
road("houston","kansas_city",120).
road("gordon","kansas_city",130).
route(Town1,Town2,Distance):-
road(Town1,Town2,Distance).
route(Town1,Town2,Distance):-
road(Town1,X,Dist1),
route(X,Town2,Dist2),
Distance=Dist1+Dist2,!.
```

Output:

```
Goal:
route("tampa", "kansas_city", X),
write("Distance from Tampa to Kansas City is ",X),nl.
Distance from Tampa to Kansas City is 320
X=320
```

5. Write a program to solve water jug problem using LISP

```
;returns the quantity in second jug
(defun get-second-jug (state) (cadr state))
;returns the state of two jugs
(defun get-state (f s) (list f s))
;checks whether a given state is a goal
; GOAL IS TO GET 4 IN SECOND JUG
(defun is-goal (state)
 (eq (get-second-jug state) 4))
;returns all possible states that can be derived
;from a given state
(defun child-states (state)
 (remove-null
(list
 (fill-first-jug state)
 (fill-second-jug state)
 (pour-first-second state)
 (pour-second-first state)
 (empty-first-jug state)
 (empty-second-jug state))))
;remove the null states
(defun remove-null (x)
 (cond
((null x) nil)
((null (car x)) (remove-null (cdr x)))
((cons (car x) (remove-null (cdr x))))))
;return the state when the first jug is filled (first jug can hold 3)
(defun fill-first-jug (state)

 (cond
((< (get-first-jug state) 3) (get-state 3 (get-second-jug state))))))
;returns the state when the second jug is filled (second jug can hold 5)
(defun fill-second-jug (state)
 (cond
((< (get-second-jug state) 5) (get-state (get-first-jug state) 5))))
;returns the state when quantity in first
;is poured to second jug
(defun pour-first-second (state)
 (let ( (f (get-first-jug state))
 (s (get-second-jug state)))
(cond
 ((zerop f) nil) ; first jug is empty
 ((= s 5) nil) ; Second jug is full
 ((<= (+ f s) 5)
 (get-state 0 (+ f s)))
 (t ; pour to first from second
 (get-state (- (+ f s) 5) 5)))))
;returns the state when second jug is poured to first
(defun pour-second-first (state)
 (let ( (f (get-first-jug state))
 (s (get-second-jug state)))
```

```
(cond
 ((zerop s) nil) ; second jug is empty
 ((= f 3) nil) ; second jug is full
 ((<= (+ f s) 3)
 (get-state (+ f s) 0))
 (t ;pour to second from first
 (get-state 3 (- (+ f s) 3))))))
;returns the state when first jug is emptied
(defun empty-first-jug (state)
 (cond
((> (get-first-jug state) 0) (get-state 0 (get-second-jug state)))))
;returns the state when second jug is emptied
(defun empty-second-jug (state)
 (cond
((> (get-second-jug state) 0) (get-state (get-first-jug state) 0))))




;;;MAIN FUNCTION
(defun dfs (start-state depth lmt)
 (setf *node* 0)
 (setf *limit* lmt)
 (dfs-node start-state depth)
)
```