Vivekanand Education Society's Institute of Technology
An Autonomous Institute Affiliated to University of Mumbai
Department of Computer Engineering

Year: 2024-2025

Name of the Course	: Artificial Intelligence (Lab)
Year/Sem/Class	: S.E.(COMP) / Sem IV / D7B
Faculty Incharge	: **Nusrat Ansari**

Code:NCMMML42

Roll No: 67	**Name:** Manish Raje	
Exp No.:	**Title:** Implement Family Tree / Tower of Hanoi / Water Jug Problem in PROLOG	
DOP:	**DOS:**	
Grade:	**Course Outcomes:**	**Signature:**

Aim: Implement Family Tree/ Water Jug Problem in PROLOG .

Theory:
Water Jug Problem:

- In the water jug problem, we are provided with two jugs; one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water.
- There is no other measuring equipment available, and the jugs also do not have any kind of marking on them.
- The agent's task here is to fill the 4-gallon jug with exactly 2 gallons of water by using only these two jugs and no other material.
- Initially, both our jugs are empty.
- To solve this problem, the following set of rules were proposed:

Production Rules for Solving the Water Jug Problem:

Here, let x denote the 4-gallon jug and y denote the 3-gallon jug.

1. (x, y) -> (4, y) // Fill the 4-gallon jug completely if x < 4
2. (x, y) -> (x, 3) // Fill the 3-gallon jug completely if y < 3
3. (x, y) -> (x-d, y) // Pour some part from the 4-gallon jug if x > 0
4. (x, y) -> (x, y-d) // Pour some part from the 3-gallon jug if y > 0
5. (x, y) -> (0, y) // Empty the 4-gallon jug if x > 0
6. (x, y) -> (x, 0) // Empty the 3-gallon jug if y > 0
7. (x, y) -> (4, y-[4-x]) // Pour some water from the 3-gallon jug to fill the 4-gallon jug if (x + y) >= 7
8. (x, y) -> (x-[3-y], y) // Pour some water from the 4-gallon jug to fill the 3-gallon jug if (x + y) >= 7
9. (x, y) -> (x+y, 0) // Pour all water from the 3-gallon jug to the 4-gallon jug if (x + y) < 4
10. (x, y) -> (0, x+y) // Pour all water from the 4-gallon jug to the 3-gallon jug if (x + y) < 3

Solution of Water Jug Problem according to the production rules:

Sr. No.	**4-gallon jug contents**	**3-gallon jug contents**	**Rule followed**
1	0 gallons	0 gallons	Initial state
2	0 gallons	3 gallons	Rule 2
3	3 gallons	0 gallons	Rule 9
4	3 gallons	3 gallons	Rule 2
5	4 gallons	2 gallons	Rule 7
6	0 gallons	2 gallons	Rule 5
7	2 gallons	0 gallons	Rule 9

On reaching the 7th attempt, we reach a state which is our goal state. Therefore, at this state, our problem is solved.

Components of Problem Formulation (with an example):

1. Initial State**:** The first component that describes the problem is the initial state that the agent starts in.
   Eg: If a taxi needs to get to location (B) but the taxi is currently at location (A), then the initial state of the problem would be location (A).
2. Actions**:**The second component that describes the problem is a description of the possible actions available to the agent. Given a state sss, Action(s) returns the set of actions that can be executed in sss. We say that each of these actions is applicable in sss.

3. Transition Model:The third component is the description of what each action does, which is called the transition model. It is specified by a function Result(s, a) that returns the state that results from doing action aaa in state sss. The initial state, actions, and transition model together define the state space of a problem, which is a set of all states reachable from the initial state by any sequence of actions. The state space forms a graph in which the nodes are states and the links between the nodes are actions.
4. Goal Test:The goal test determines whether a given state is a goal state or not. Sometimes there is an explicit list of possible goal states and the test simply checks whether the given state is one of them; sometimes the goal is specified by an abstract property rather than an explicitly enumerated set of states.
5. Path Cost:The last component of the problem is the path cost, which is a function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure. The solution to the problem is an action sequence that leads from the initial state to the goal state, and the solution quality is measured by the path cost function. An optimal solution is the lowest path cost among all the solutions.

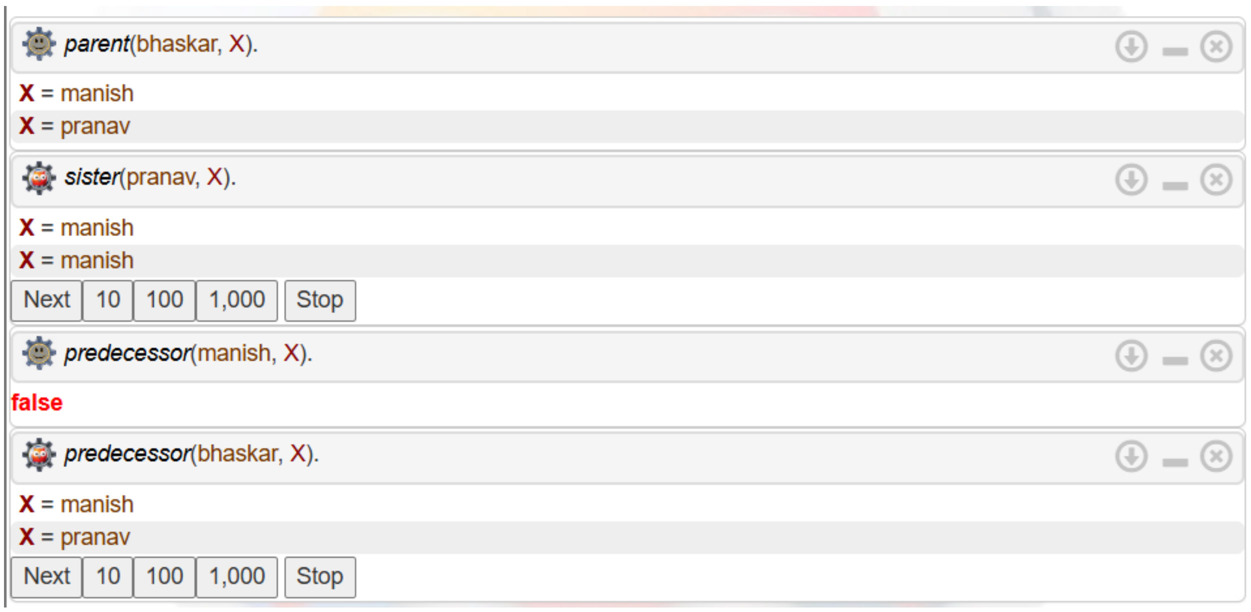
FAMILY TREE

```
parent(bhaskar, manish).

parent(bhaskar, pranav).

parent(usha, manish).

parent(usha, pranav).

parent(govind, bhaskar).

parent(lata, bhaskar).

parent(govind, chandrakant).

parent(ramprasad, govind).


female(pranav).

female(usha).

female(lata).
```

```
female(sita).

male(manish).

male(bhaskar).

male(govind).

male(chandrakant).


offspring(Y, X):-
    parent(X, Y).


mother(X, Y):-
    parent(X, Y),
    female(X).
```

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

sister(X,Y):-
    parent(Z,X),
    parent(Z,Y),
    female(X),
    X \= Y.

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

OUTPUT



```
parent(bhaskar, X).
X = manish
X = pravav
```

```
sister(pranav, X).
X = manish
X = manish
Next 10 100 1,000 Stop
```

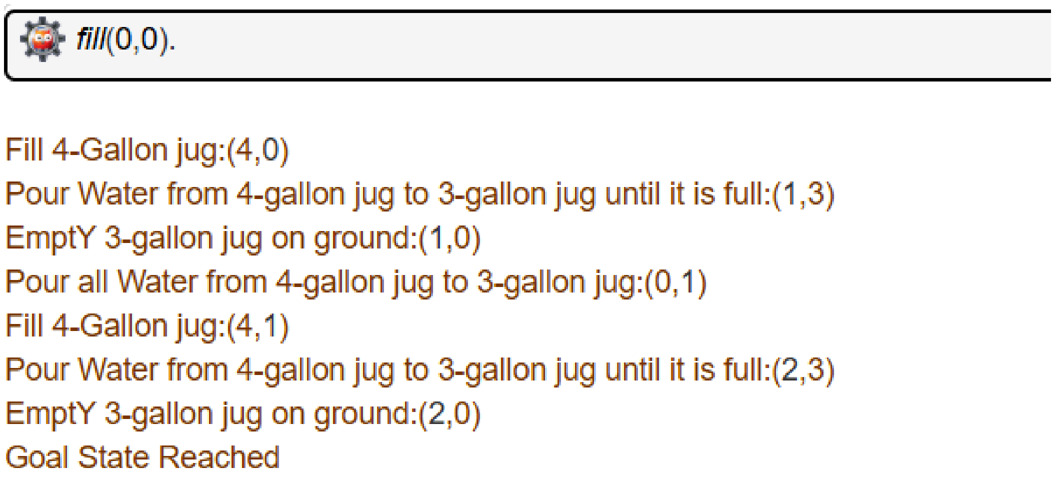
```
predecessor(manish, X).
false
```

```
predecessor(bhaskar, X).
X = manish
X = pranav
Next 10 100 1,000 Stop
```

WATER JUG PROBLEM

```
fill(2,0) :-
    nl,
    write('Goal State Reached').

fill(X,Y) :-
    X = 0, Y =< 1,
    nl,
    write('Fill 4-Gallon jug: (4,'),
    write(Y),
    write(')'),
    fill(4,Y).

fill(X,Y) :-
    Y = 0, X = 3,
    nl,
    write('Fill 3-Gallon jug: ('),
    write(X),
    write(',3)'),
    fill(X,3).

fill(X,Y) :-
    X + Y >= 4, Y = 3, X = 3,
    Y1 is Y - (4 - X),
    nl,
    write('Pour Water from 3-gallon jug to 4-
gallon jug until it is full: (4,'),
    write(Y1),
    write(')'),
    fill(4,Y1).

fill(X,Y) :-
    X + Y >= 3, Y =< 1, X = 4,
    X1 is X - (3 - Y),
    nl,
    write('Pour Water from 4-gallon jug to 3-
gallon jug until it is full: ('),
    write(X1),
    write(',3)'),
    fill(X1,3).

fill(X,Y) :-
    X + Y =< 4, X = 0, Y > 1,
    X1 is X + Y,
    nl,
    write('Pour all Water from 3-gallon jug to
4-gallon jug: ('),
    write(X1),
    write(',0)'),
    fill(X1,0).

fill(X,Y) :-
    X + Y < 3, Y = 0,
    Y1 is X + Y,
    nl,
    write('Pour all Water from 4-gallon jug to
3-gallon jug: (0,'),
    write(Y1),
    write(')'),
    fill(0,Y1).

fill(X,Y) :-
    Y = 2, X = 4,
    nl,
    write('Empty 4-gallon jug on ground:
(0,'),
    write(Y),
    write(')'),
    fill(0,Y).

fill(X,Y) :-
    Y = 3, X >= 1,
    nl,
    write('Empty 3-gallon jug on ground: ('),
    write(X),
    write(',0)'),
    fill(X,0).

fill(X,Y) :-
    X > 4, Y < 3,
    write('4-gallon jug overflow:'), nl.

fill(X,Y) :-
    X < 4, Y > 3,
    write('3-gallon jug overflow:'), nl.

fill(X,Y) :-
    X > 4, Y > 3,
    write('4-gallon jug and 3-gallon jug
overflow:'), nl.
```

OUTPUT



```
fill(0,0).

Fill 4-Gallon jug:(4,0)
Pour Water from 4-gallon jug to 3-gallon jug until it is full:(1,3)
EmptY 3-gallon jug on ground:(1,0)
Pour all Water from 4-gallon jug to 3-gallon jug:(0,1)
Fill 4-Gallon jug:(4,1)
Pour Water from 4-gallon jug to 3-gallon jug until it is full:(2,3)
EmptY 3-gallon jug on ground:(2,0)
Goal State Reached
```