% Define jug capacities

capacity(4, 3).

% Goal: we want 2 liters in either jug

goal(state(2, \_)).

goal(state(\_, 2)).

% Moves with description for each

move(state(\_, J2), state(4, J2), 'Fill the 4-Gallon Jug').

move(state(J1, \_), state(J1, 3), 'Fill the 3-Gallon Jug').

move(state(\_, J2), state(0, J2), 'Empty the 4-Gallon Jug on ground').

move(state(J1, \_), state(J1, 0), 'Empty the 3-Gallon Jug on ground').

move(state(J1, J2), state(NewJ1, NewJ2), 'Pour water from 4-Gallon Jug to 3-Gallon Jug') :-

J1 > 0, J2 < 3, Pour is min(J1, 3 - J2),

NewJ1 is J1 - Pour, NewJ2 is J2 + Pour.

move(state(J1, J2), state(NewJ1, NewJ2), 'Pour water from 3-Gallon Jug to 4-Gallon Jug') :-

J2 > 0, J1 < 4, Pour is min(J2, 4 - J1),

NewJ1 is J1 + Pour, NewJ2 is J2 - Pour.

% Display each step with a message and check for repeated states

solve\_dfs(state(J1, J2), Visited, []) :-

goal(state(J1, J2)),

write('Goal reached: (2 gallons in one of the jugs)'), nl.

solve\_dfs(state(J1, J2), Visited, [MoveDescription | RestMoves]) :-

move(state(J1, J2), state(NewJ1, NewJ2), MoveDescription),

\+ member(state(NewJ1, NewJ2), Visited), % Ensure the state hasn't been visited

write(MoveDescription), write(': ('), write(J1), write(', '), write(J2), write(') --> ('),

write(NewJ1), write(', '), write(NewJ2), write(')'), nl,

solve\_dfs(state(NewJ1, NewJ2), [state(NewJ1, NewJ2) | Visited], RestMoves).

% Start solving from (0, 0) and display a title

solve :-

write("+-----------------------------4-3 Water Jug Problem--------------------------+"), nl,

solve\_dfs(state(0, 0), [state(0, 0)], \_).

% Automatically run the solver

:- initialization(solve).