```
% water_jug(StartJug1, StartJug2, Target, Solution) solves the water jug problem where:
% - StartJug1 and StartJug2 are the initial water amounts in both jugs,
% - Target is the target amount of water to measure,
% - Solution is a list of steps showing the process to reach the target.
water_jug(StartJug1, StartJug2, Target, Solution) :-
    solve(StartJug1, StartJug2, Target, [], Solution).
% solve(Amount1, Amount2, Target, History, Solution)
% This is the recursive helper predicate that tries all possible moves.
solve(Amount1, Amount2, Target, History, Solution) :-
    Amount1 = Target, % if jug 1 has the target amount
    reverse(History, Solution).
solve(Amount1, Amount2, Target, History, Solution) :-
    Amount2 = Target, % if jug 2 has the target amount
    reverse(History, Solution).
solve(Amount1, Amount2, Target, History, Solution) :-
    % If not reached the target, try all possible actions
     % Fill jug 1 completely
    fill_jug1(Amount1, Amount2, NewAmount1, NewAmount2),
    \+ member([NewAmount1, NewAmount2], History),
    solve(NewAmount1, NewAmount2, Target, [[NewAmount1, NewAmount2] | History], Solution).
    solve(Amount1, Amount2, Target, History, Solution) :-
    % Fill jug 2 completely
    fill_jug2(Amount1, Amount2, NewAmount1, NewAmount2),
    \+ member([NewAmount1, NewAmount2], History),
    solve(NewAmount1, NewAmount2, Target, [[NewAmount1, NewAmount2] | History], Solution).
    solve(Amount1, Amount2, Target, History, Solution) :-
    % Empty jug 1
```

```
 empty_jug1(Amount1, Amount2, NewAmount1, NewAmount2),
   \+ member([NewAmount1, NewAmount2], History),
   solve(NewAmount1, NewAmount2, Target, [[NewAmount1, NewAmount2] | History], Solution).
   solve(Amount1, Amount2, Target, History, Solution) :-
   % Empty jug 2
   empty_jug2(Amount1, Amount2, NewAmount1, NewAmount2),
   \+ member([NewAmount1, NewAmount2], History),
   solve(NewAmount1, NewAmount2, Target, [[NewAmount1, NewAmount2] | History], Solution).
   solve(Amount1, Amount2, Target, History, Solution) :-
   % Pour from jug 1 to jug 2
   pour_jug1_to_jug2(Amount1, Amount2, NewAmount1, NewAmount2),
   \+ member([NewAmount1, NewAmount2], History),
   solve(NewAmount1, NewAmount2, Target, [[NewAmount1, NewAmount2] | History], Solution).
   solve(Amount1, Amount2, Target, History, Solution) :-
   % Pour from jug 2 to jug 1
   pour_jug2_to_jug1(Amount1, Amount2, NewAmount1, NewAmount2),
   \+ member([NewAmount1, NewAmount2], History),
   solve(NewAmount1, NewAmount2, Target, [[NewAmount1, NewAmount2] | History], Solution).
% Actions that can be performed:
% Fill jug 1
fill_jug1(_, Amount2, Jug1, Amount2) :- Jug1 = 5.  % assuming jug 1's capacity is 5 liters
% Fill jug 2
fill_jug2(Amount1, _, Amount1, Jug2) :- Jug2 = 3.  % assuming jug 2's capacity is 3 liters
% Empty jug 1
empty_jug1(_, Amount2, 0, Amount2).
% Empty jug 2
empty_jug2(Amount1, _, Amount1, 0).
```

```
% Pour from jug 1 to jug 2
pour_jug1_to_jug2(Amount1, Amount2, NewAmount1, NewAmount2) :-
    Pour = min(Amount1, 3 - Amount2),
    NewAmount1 is Amount1 - Pour,
    NewAmount2 is Amount2 + Pour.
% Pour from jug 2 to jug 1
pour_jug2_to_jug1(Amount1, Amount2, NewAmount1, NewAmount2) :-
    Pour = min(Amount2, 5 - Amount1),
    NewAmount1 is Amount1 + Pour,
    NewAmount2 is Amount2 - Pour.
```