Human And Monkey :Let's cross the river

1. Three humans, one big Monkey and two small monkeys are to cross a river:
2. Only humans and the big monkey can row the boat.
3. At all times, the number of humans on either side of the river must be greater or equal to the number of monkeys on that side. (Or else the humans will be eaten by the monkeys!)
4. The boat only has room for 2 (monkeys or humans).
5. Monkeys can jump out of the boat when it's banked.

Solution :

Main Logic

The target of the game is to transfer all the 3 Humans , 1 Big monkey and 2 monkeys to the other side of the river. So we can represent the state of the two sides of the river and the boat in form of a tuple. Let the tuple be like :

[HL,BML,ML,S,HR,BMR,MR]

Where
HR -> No of Humans on right side of river
HL -> No of Humans on left side of river
BML -> No of Big monkey on left side of river
BMR -> No of Big monkey on right side of river
ML -> No of Monkey on left side of river
MR -> No of Monkey on right side of river
S -> The side of the river on which boat is situated

The tuple is represented as a list.

For the tuple to be valid we have to check the following conditions :

1. **No of Humans on a side >=0 & <=3**
2. **No of Monkeys on a side >=0 & <=2**
3. **No of Big Monkeys on a side>=0 & <=1**
4. **No of Humans on a side >= No of Monkeys + No of Big monkeys OR No of Humans is 0**

The above conditions can be checked using the possible() predicate :

```
possible(HL,BML,ML,HR,BMR,MR) :-
    HL>=0, BML>=0, ML>=0,
    HL=<3, BML=<1, ML=<2,
    HR>=0, BMR>=0, MR>=0,
    HR=<3, BMR=<1, MR=<2,
    MonkeysOnLeft is ML+BML,
    MonkeysOnRight is MR+BMR,
    HumansOnRight is HR,
    HumansOnLeft is HL,
    (HumansOnLeft>=MonkeysOnLeft ; HumansOnLeft==0),
    (HumansOnRight>=MonkeysOnRight ; HumansOnRight==0).
```

Now we have to change the state when the Human and the monkeys travel in the boat. We have to change the state according to the members in the boat. I did that in the program using the row() function. There is a total of 6 possibilities from left to right and vice versa . They are :

From left to right

----

1. **Two humans cross left to right**
2. **One big monkey and one monkey cross left to right**
3. **One Human and One monkey cross left to right**
4. **One Human and One big monkey left to right**
5. **One human cross left to right**
6. **One Big monkey cross left to right**

From right to left

----

1. **Two humans cross right to left**
2. **One big monkey and one monkey cross right to left**
3. **One Human and One monkey cross right to left**
4. **One Human and One big monkey right to left**
5. **One human cross right to left**
6. **One Big monkey cross right to left**

An example is shown below. If two humans are travelling from left to right then the state changes such that no humans in left side in the new state is reduced by 2 and no of humans on the right side in the new state is increased by 2. In the Final step the we check whether the state is valid using the possible() predicate.

```
row([HL,BML,ML,left,HR,BMR,MR],[HL1,BML,ML,right,HR1,BMR,MR]):-
    %% 1. Two humans cross left to right
    HR1 is HR+2,
    HL1 is HL-2,
    possible(HL1,BML,ML,HR1,BMR,MR).
```

Now coming to the part where we solve the problem. In the solution I use the Solve() functor which accepts 4 arguments as follows:

1. **Current state - Contains a tuple showing the current state of river**
2. **Goal state - Contains the Goal state we have to achieve**
3. **Visited List - Contains the states which we already have tried.**
4. **History List - Contains a List of List of two elements each to show all the States we have traversed.**

How solve() works :

- **It will traverse through all the moves defined**
- **Then we check the new state produced by the new move is already in the visited array or not. By using(not() and member())**
- **Then we will make the new state as current state and recursively search for the goal state.**

The solve function is as follows :

```
solve([HL1,BML1,ML1,S1,HR1,BMR1,MR1],[HL2,BML2,ML2,S2,HR2,BMR2,MR2],Visited,History) :-
row([HL1,BML1,ML1,S1,HR1,BMR1,MR1],[HL3,BML3,ML3,S3,HR3,BMR3,MR3]),
not(member([HL3,BML3,ML3,S3,HR3,BMR3,MR3],Visited)),
solve([HL3,BML3,ML3,S3,HR3,BMR3,MR3],[HL2,BML2,ML2,S2,HR2,BMR2,MR2],[[HL3,BML3,ML3,S3,HR3,BMR3,MR3]|Visited],[
[[HL3,BML3,ML3,S3,HR3,BMR3,MR3],[HL1,BML1,ML1,S1,HR1,BMR1,MR1]] | History ]).
solve([HL,BML,ML,S,HR,BMR,MR],[HL,BML,ML,S,HR,BMR,MR],_,History):-
 printResult(History).
```

Printing the result is done as follows :

1. **Recursively traverse till the end of the History array**
2. **The second element in the list is the Initial state and the first element is the Final state**
3. **Print using the format() function**

```
printResult([]) :- format('~n').
printResult([[[HL1,BML1,ML1,S1,HR1,BMR1,MR1],[HL,BML,ML,S,HR,BMR,MR]]|History]) :-
    printResult(History),
     format(' ~w  ~w ,~w  ~w  ~w ,~w  ',[HL,BML,ML,HR,BMR,MR]),nl,
     format(' ~w  ~w ,~w   ~w  ~w ,~w  ',[HL1,BML1,ML1,HR1,BMR1,MR1]),nl.
```

Input is given as follows :

```
find :-
solve([3,1,2,left,0,0,0],[0,0,0,right,3,1,2],[[3,1,2,left,0,0,0]],_).
```

solve([3,1,2,left,0,0,0],[0,0,0,right,3,1,2],[[3,1,2,left,0,0,0]],_).

[3,1,2,left,0,0,0] is the initial state and [0,0,0,right,3,1,2] is the final state. Initially we add the Initial state to the Visited list as we already visited it.

Output

```
?- find.
            Side1                  ---------------------------        Side2

 3 Humans, 1 Big Monkey ,2 Monkeys ----------river---------- 0 Humans, 0 Big Monkey ,0 Monkeys
 3 Humans, 0 Big Monkey ,1 Monkeys ----------river---------- 0 Humans, 1 Big Monkey ,1 Monkeys
 3 Humans, 0 Big Monkey ,1 Monkeys ----------river---------- 0 Humans, 1 Big Monkey ,1 Monkeys
 3 Humans, 1 Big Monkey ,1 Monkeys ----------river---------- 0 Humans, 0 Big Monkey ,1 Monkeys
 3 Humans, 1 Big Monkey ,1 Monkeys ----------river---------- 0 Humans, 0 Big Monkey ,1 Monkeys
 3 Humans, 0 Big Monkey ,0 Monkeys ----------river---------- 0 Humans, 1 Big Monkey ,2 Monkeys
 3 Humans, 0 Big Monkey ,0 Monkeys ----------river---------- 0 Humans, 1 Big Monkey ,2 Monkeys
 3 Humans, 1 Big Monkey ,0 Monkeys ----------river---------- 0 Humans, 0 Big Monkey ,2 Monkeys
 3 Humans, 1 Big Monkey ,0 Monkeys ----------river---------- 0 Humans, 0 Big Monkey ,2 Monkeys
 1 Humans, 1 Big Monkey ,0 Monkeys ----------river---------- 2 Humans, 0 Big Monkey ,2 Monkeys
 1 Humans, 1 Big Monkey ,0 Monkeys ----------river---------- 2 Humans, 0 Big Monkey ,2 Monkeys
 2 Humans, 1 Big Monkey ,1 Monkeys ----------river---------- 1 Humans, 0 Big Monkey ,1 Monkeys
 2 Humans, 1 Big Monkey ,1 Monkeys ----------river---------- 1 Humans, 0 Big Monkey ,1 Monkeys
 1 Humans, 0 Big Monkey ,1 Monkeys ----------river---------- 2 Humans, 1 Big Monkey ,1 Monkeys
 1 Humans, 0 Big Monkey ,1 Monkeys ----------river---------- 2 Humans, 1 Big Monkey ,1 Monkeys
 2 Humans, 0 Big Monkey ,2 Monkeys ----------river---------- 1 Humans, 1 Big Monkey ,0 Monkeys
 2 Humans, 0 Big Monkey ,2 Monkeys ----------river---------- 1 Humans, 1 Big Monkey ,0 Monkeys
 0 Humans, 0 Big Monkey ,2 Monkeys ----------river---------- 3 Humans, 1 Big Monkey ,0 Monkeys
 0 Humans, 0 Big Monkey ,2 Monkeys ----------river---------- 3 Humans, 1 Big Monkey ,0 Monkeys
 0 Humans, 1 Big Monkey ,2 Monkeys ----------river---------- 3 Humans, 0 Big Monkey ,0 Monkeys
 0 Humans, 1 Big Monkey ,2 Monkeys ----------river---------- 3 Humans, 0 Big Monkey ,0 Monkeys
 0 Humans, 0 Big Monkey ,1 Monkeys ----------river---------- 3 Humans, 1 Big Monkey ,1 Monkeys
 0 Humans, 0 Big Monkey ,1 Monkeys ----------river---------- 3 Humans, 1 Big Monkey ,1 Monkeys
 1 Humans, 0 Big Monkey ,1 Monkeys ----------river---------- 2 Humans, 1 Big Monkey ,1 Monkeys
 1 Humans, 0 Big Monkey ,1 Monkeys ----------river---------- 2 Humans, 1 Big Monkey ,1 Monkeys
 0 Humans, 0 Big Monkey ,0 Monkeys ----------river---------- 3 Humans, 1 Big Monkey ,2 Monkeys
true
```