# AI PROLOG ASSIGNMENT 2
# IIT2016128

## Ques1: Human Monkey river problem.

### Problem Statement:

Human And Monkey: Let's cross the river
Three humans, one big Monkey and two small monkeys are to cross a river:

1. Only humans and the big monkey can row the boat.
2. 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!)
3. The boat only has room for 2 (monkeys or humans).
4. Monkeys can jump out of the boat when it's banked.

### Solution Approach

The given problem can be solved by the most common graph algorithms namely breadth first search (dfs) or depth first search (dfs). Since prolog is logic programming language and based on recursion (stack based), it is good to implement dfs algorithm.

The program includes 3 basic functions:

1. is_ok
2. move_from
3. state

4th is a caller function solution and last 5th function is print function printanswer.

### Variable description:

1. **H**: refers to humans.
2. **SM**: refers to small monkey.
3. **BM:** refers to big monkey.
4. **L**: refers to left.
5. **R:** refers to right.

   **For example**: **HL** refers to numbers of humans on left side of river and **SMR** refers to number of monkeys on right side of river.

### Function descriptions:

1. is_ok
   is_ok is used to check at if any given state is valid or not. It is possible that while trying for moves, the no. of humans or monkeys exceeds or decreases by the threshold value. To make sure that doesn't happens, after every move the state is checked if it is a valid or not.

```
is_ok(HL,SML,BML,HR,SMR,BMR) :-
    HL >= 0, SML >= 0, BML >= 0, HR >= 0, SMR >= 0, BMR >= 0,
    HL =< 3, SML =< 2, BML =< 1, HR =< 3, SMR =< 2, BMR =< 1,
    (HL >= SML + BML ; HL = 0),
    (HR >= SMR + BMR ; HR = 0).
```

2. move_from
   move_from function is used to generate a new move state from the input state and check if it is a valid state or not using is_ok function. These moves will be the one which will lead to the solution. For the given problem there are 12 possible moves as listed below:
   1. one big and one small monkey moves from left to right
   2. one big and one small monkey moves from right to left
   3. two humans move from left to right
   4. two humans move from right to left
   5. one human and one big monkey moves from left to right
   6. one human and one big monkey moves from right to left
   7. one human and one small monkey moves from left to right
   8. one human and one small monkey moves from right to left
   9. one human moves from left to right
   10. one human moves from right to left
   11. one big monkey moves from right to left
   12. one big monkey moves from left to right

```
% two humans move from left to right
move_from([HL, SML, BML, HR, SMR, BMR, left], [HL2, SML, BML, HR2, SMR, BMR, right]):-
    HL2 is HL - 2,
    HR2 is HR + 2,
    is_ok(HL2, SML, BML, HR2, SMR, BMR).

% two humans move from right to left
move_from([HL, SML, BML, HR, SMR, BMR, right], [HL2, SML, BML, HR2, SMR, BMR, left]):-
    HL2 is HL + 2,
    HR2 is HR - 2,
    is_ok(HL2, SML, BML, HR2, SMR, BMR).
```

3. state
   Arguments: current state
   goal state
   visited (states visited till now)
   list_of_moves (a list of moves which will take us to the solution, this will be the answer)
   state function is used to solve the main problem. It generates all the possible moves from the current input state, after that it checks if it is currently in the visited list or not, if it is not

in the visited list then it must be visited as no new paths are currently known from that state. Finally it is recursively called again and again with the new state generated from Move_from until the goal state is reached. This implements dfs algorithm.

```
state([HL,SML,BML,HR,SMR,BMR,Boat], [HL1,SML1,BML1,HR1,SMR1,BMR1,Boat1], Visited, List_of_moves):-
    move_from([HL,SML,BML,HR,SMR,BMR,Boat], [HL2,SML2,BML2,HR2,SMR2,BMR2,Boat2]),
    not(member([HL2,SML2,BML2,HR2,SMR2,BMR2,Boat2], Visited)),
    state([HL2,SML2,BML2,HR2,SMR2,BMR2,Boat2], [HL1,SML1,BML1,HR1,SMR1,BMR1,Boat1], [[HL2,SML2,BML2,HR2,SMR2,BMR2,Boat2]|Visited], [[[HL2,SML2,BML2,HR2,SMR2,BMR2,Boat2], [HL


state([0,0,0,3,2,1,right], [0,0,0,3,2,1,right], _, List_of_moves):-
    printanswer(List_of_moves).
```

4. printanswer
   This function takes the list_of_moves generated from the state and prints it.

```
printanswer([]):- nl.
printanswer([[A,B] | Moves]):-
    printanswer(Moves),
    write(' Move from '), write(B), write(' --> '), write(' to '), write(A), nl.
```

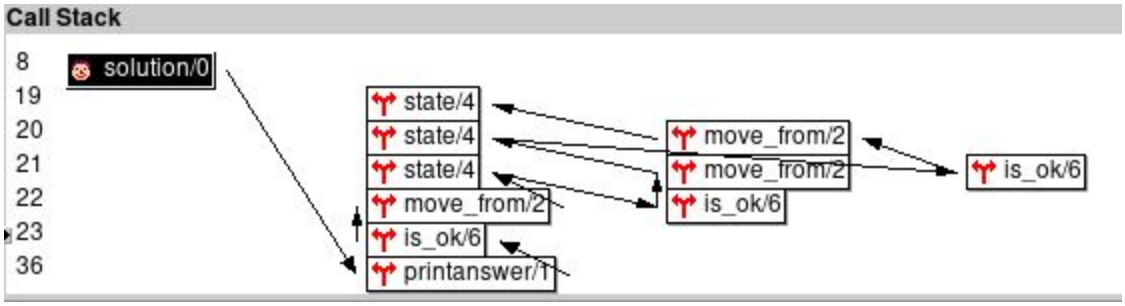
5. solution
   This is the main caller function which is called to get the solution.

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

## Algorithm:

1. The program uses dfs (depth first search algorithm) to search for the required goal states from different states generated from move_from function.
2. The input state is all the 3 monkeys and humans on the left side of the river. From the given input state, the possible valid moves are generated and firstly checked if they are visited or not, if they are not visited then they are put up in the visited list and that state become as the input state for next call of the function.
3. In this way the possible moves are generated and checked. If at any point a visited state is encountered then that path will not lead to answer and other states are being checked for the solution.
4. At each step the valid possible state is added to the list_of_moves which stores the states leading to the solution.

## Function call stack (taken from gtracer of prolog):




## Running Instructions:

Load the program file and then query “solution”.

## Code Output: