## Learning Abstract:

This assignment is made up nine tasks which involve a mixture of using premade code in addition to deducing my own predicates to solve the classic Towers of Hanoi problem but in a Prolog context. In addition to this, this assignment is intended to further develop a conceptual understanding of state space problem solving. The demos included are for the 3- and 4-disc part of the problem. An interesting worthwhile side artifact is that when I attempted to run the problem as a 5-disc problem the program was not able to finitely define a solution. This is likely due in part to the fact that program is performing a blind search in which adding an additional disk yield in exponentially longer compute times with each addition of a disc. Different search methodologies could solve this problem .

## Task 1:

Contemplate the nature of the problem, see specification on web page for details.

## Task 2:

Copy and paste source code and check to ensure validity and that it initially compiles, see specification on web page for details, full code posted later this document.

## Task 3: One Move Predicate and a Unit Test

m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-

    Tower1Before = [H|T],

    Tower1After = T,

    Tower2Before = L,

    Tower2After = [H|L].

test\_\_m12 :-

    write('Testing: move\_m12\n'),

    TowersBefore = [[t,s,m,l,h],[],[]],

    trace('','TowersBefore',TowersBefore),

    m12(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

Text

Description automatically generated

## Task 4: The Remaining Five Move Predicates and a Unit Tests

m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-

    Tower1Before = [H|T],

    Tower1After = T,

    Tower2Before = L,

    Tower2After = [H|L].

m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-

    Tower1Before = [H|T],

    Tower1After = T,

    Tower3Before = L,

    Tower3After = [H|L].

m21([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-

    Tower2Before = [H|T],

    Tower2After = T,

    Tower1Before = L,

    Tower1After = [H|L].

m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-

    Tower2Before = [H|T],

    Tower2After = T,

    Tower3Before = L,

    Tower3After = [H|L].

m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-

    Tower3Before = [H|T],

    Tower3After = T,

    Tower1Before = L,

    Tower1After = [H|L].

m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-

    Tower3Before = [H|T],

    Tower3After = T,

    Tower2Before = L,

    Tower2After = [H|L].

% --- Unit test programs

test\_\_m12 :-

    write('Testing: move\_m12\n'),

    TowersBefore = [[t,s,m,l,h],[],[]],

    trace('','TowersBefore',TowersBefore),

    m12(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m13 :-

    write('Testing: move\_m13\n'),

    TowersBefore = [[t,s,m,l,h],[],[]],

    trace('','TowersBefore',TowersBefore),

    m13(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m21 :-

    write('Testing: move\_m21\n'),

    TowersBefore = [[],[t,s,m,l,h],[]],

    trace('','TowersBefore',TowersBefore),

    m21(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m23 :-

    write('Testing: move\_m23\n'),

    TowersBefore = [[],[t,s,m,l,h],[]],

    trace('','TowersBefore',TowersBefore),

    m23(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m31 :-

    write('Testing: move\_m31\n'),

    TowersBefore = [[],[],[t,s,m,l,h]],

    trace('','TowersBefore',TowersBefore),

    m31(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m32 :-

    write('Testing: move\_m32\n'),

    TowersBefore = [[],[],[t,s,m,l,h]],

    trace('','TowersBefore',TowersBefore),

    m32(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

A screenshot of a computer

Description automatically generated with medium confidence

## Task 5: Valid State Predicate and Unit Test

% -----------------------------------------------------------------------

% --- valid\_state(S) :: S is a valid state

valid\_state([A|[B|[C]]]) :- towerState(A), towerState(B), towerState(C).

towerState([]).

towerState([s]).

towerState([s,m]).

towerState([s,m,l]).

towerState([s,l]).

towerState([s,l,h]).

towerState([s,h]).

towerState([s,m,h]).

towerState([m]).

towerState([m,l]).

towerState([m,l,h]).

towerState([m,h]).

towerState([m]).

towerState([m,l]).

towerState([m,l,h]).

towerState([m,h]).

towerState([l]).

towerState([l,h]).

towerState([h]).

towerState([s,m,l,h]).

towerState([t]).

towerState([t,s]).

towerState([t,s,m]).

towerState([t,s,m,l]).

towerState([t,s,l]).

towerState([t,s,l,h]).

towerState([t,s,h]).

towerState([t,s,m,h]).

towerState([t,m]).

towerState([t,m,l]).

towerState([t,m,l,h]).

towerState([t,m,h]).

towerState([t,m]).

towerState([t,m,l]).

towerState([t,m,l,h]).

towerState([t,m,h]).

towerState([t,l]).

towerState([t,l,h]).

towerState([t,h]).

towerState([t,s,m,l,h]).

%% Unit Test Code

test\_\_valid\_state :-

    write('Testing: valid\_state\n'),

    test\_\_vs([[l,t,s,m,h],[],[]]),

    test\_\_vs([[t,s,m,l,h],[],[]]),

    test\_\_vs([[],[h,t,s,m],[l]]),

    test\_\_vs([[],[t,s,m,h],[l]]),

    test\_\_vs([[],[h],[l,m,s,t]]),

    test\_\_vs([[],[h],[t,s,m,l]]).

test\_\_vs(S) :-

    valid\_state(S),

    write(S), write(' is valid.'), nl.

test\_\_vs(S) :-

    write(S), write(' is invalid.'), nl.

Text

Description automatically generated

## Task 6: Defining the write sequence predicate

%% Write Sequence Doe

write\_sequence([]).

write\_sequence([H|T]) :-

    elaborate(H,E),

    write(E),nl,

    write\_sequence(T).

elaborate(m12,Output) :-

    Output = 'Transfer a disk from tower 1 to tower 2.'.

elaborate(m13,Output) :-

    Output = 'Transfer a disk from tower 1 to tower 3.'.

elaborate(m21,Output) :-

    Output = 'Transfer a disk from tower 2 to tower 1.'.

elaborate(m23,Output) :-

    Output = 'Transfer a disk from tower 2 to tower 3.'.

elaborate(m31,Output) :-

    Output = 'Transfer a disk from tower 3 to tower 1.'.

elaborate(m32,Output) :-

    Output = 'Transfer a disk from tower 3 to tower 2.'.

%% Unit Test Code

test\_\_write\_sequence :-

    write('First test of write\_sequence ...'), nl,

    write\_sequence([m31,m12,m13,m21]),

    write('Second test of write\_sequence ...'), nl,

    write\_sequence([m13,m12,m32,m13,m21,m23,m13]).

Text

Description automatically generated

## Task 7: Intermediate and Plain English Demo

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

#### Paraphrased English Solution

Calendar

Description automatically generated with medium confidence

### Questions and Answers

#### What was the length of your program’s solutions to the three-disk problem?

##### The length appears to be a degree of 14 steps.

#### What is the length of the shortest solution to the three-disk problem?

##### Doing it by hand I was able to complete it in 7 steps with three discs.

#### How do you account for the discrepancy?

##### It appears the program is checking each possible state before executing another transition bringing into question the computational efficiency of this process.

#### Task 8:

A picture containing text, computer

Description automatically generated

### Questions and Answers

#### What was the length of your program’s solutions to the four-disk problem?

##### The length appears to be a degree of 40 steps.

#### What is the length of the shortest solution to the four-disk problem?

##### Doing it by hand I was able to complete it in 15 steps with four discs.

## Task 9: The Full Code Base

% -----------------------------------------------------------------------

% -----------------------------------------------------------------------

% --- File: towers\_of\_hanoi.pro

% --- Line: Program to solve the Towers of Hanoi problem

% -----------------------------------------------------------------------

:- consult('inspectors.pro').

% -----------------------------------------------------------------------

% --- make\_move(S,T,SSO) :: Make a move from state S to state T by SSO

make\_move(TowersBeforeMove,TowersAfterMove,m12) :-

    m12(TowersBeforeMove,TowersAfterMove).

make\_move(TowersBeforeMove,TowersAfterMove,m13) :-

    m13(TowersBeforeMove,TowersAfterMove).

make\_move(TowersBeforeMove,TowersAfterMove,m21) :-

    m21(TowersBeforeMove,TowersAfterMove).

make\_move(TowersBeforeMove,TowersAfterMove,m23) :-

    m23(TowersBeforeMove,TowersAfterMove).

make\_move(TowersBeforeMove,TowersAfterMove,m31) :-

    m31(TowersBeforeMove,TowersAfterMove).

make\_move(TowersBeforeMove,TowersAfterMove,m32) :-

    m32(TowersBeforeMove,TowersAfterMove).

m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-

    Tower1Before = [H|T],

    Tower1After = T,

    Tower2Before = L,

    Tower2After = [H|L].

m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-

    Tower1Before = [H|T],

    Tower1After = T,

    Tower3Before = L,

    Tower3After = [H|L].

m21([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-

    Tower2Before = [H|T],

    Tower2After = T,

    Tower1Before = L,

    Tower1After = [H|L].

m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-

    Tower2Before = [H|T],

    Tower2After = T,

    Tower3Before = L,

    Tower3After = [H|L].

m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-

    Tower3Before = [H|T],

    Tower3After = T,

    Tower1Before = L,

    Tower1After = [H|L].

m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-

    Tower3Before = [H|T],

    Tower3After = T,

    Tower2Before = L,

    Tower2After = [H|L].

% -----------------------------------------------------------------------

% --- valid\_state(S) :: S is a valid state

valid\_state([A|[B|[C]]]) :- towerState(A), towerState(B), towerState(C).

towerState([]).

towerState([s]).

towerState([s,m]).

towerState([s,m,l]).

towerState([s,l]).

towerState([s,l,h]).

towerState([s,h]).

towerState([s,m,h]).

towerState([m]).

towerState([m,l]).

towerState([m,l,h]).

towerState([m,h]).

towerState([m]).

towerState([m,l]).

towerState([m,l,h]).

towerState([m,h]).

towerState([l]).

towerState([l,h]).

towerState([h]).

towerState([s,m,l,h]).

towerState([t]).

towerState([t,s]).

towerState([t,s,m]).

towerState([t,s,m,l]).

towerState([t,s,l]).

towerState([t,s,l,h]).

towerState([t,s,h]).

towerState([t,s,m,h]).

towerState([t,m]).

towerState([t,m,l]).

towerState([t,m,l,h]).

towerState([t,m,h]).

towerState([t,m]).

towerState([t,m,l]).

towerState([t,m,l,h]).

towerState([t,m,h]).

towerState([t,l]).

towerState([t,l,h]).

towerState([t,h]).

towerState([t,s,m,l,h]).

% -----------------------------------------------------------------------

% --- solve(Start,Solution) :: succeeds if Solution represents a path

% --- from the start state to the goal state.

solve :-

    extend\_path([[[s,m,l,h],[],[]]],[],Solution),

    write\_solution(Solution).

extend\_path(PathSoFar,SolutionSoFar,Solution) :-

    PathSoFar = [[[],[],[s,m,l,h]]|\_],

    % showr('PathSoFar',PathSoFar),

    % showr('SolutionSoFar',SolutionSoFar),

    Solution = SolutionSoFar.

extend\_path(PathSoFar,SolutionSoFar,Solution) :-

    PathSoFar = [CurrentState|\_],

    % showr('PathSoFar',PathSoFar),

    make\_move(CurrentState,NextState,Move),

    % show('Move',Move),

    % show('NextState',NextState),

    not(member(NextState,PathSoFar)),

    valid\_state(NextState),

    Path = [NextState|PathSoFar],

    Soln = [Move|SolutionSoFar],

    extend\_path(Path,Soln,Solution).

% -----------------------------------------------------------------------

% --- write\_sequence\_reversed(S) :: Write the sequence, given by S,

% --- expanding the tokens into meaningful strings.

write\_solution(S) :-

    nl, write('Solution ...'), nl, nl,

    reverse(S,R),

    write\_sequence(R),nl.

write\_sequence([]).

write\_sequence([H|T]) :-

    elaborate(H,E),

    write(E),nl,

    write\_sequence(T).

elaborate(m12,Output) :-

    Output = 'Transfer a disk from tower 1 to tower 2.'.

elaborate(m13,Output) :-

    Output = 'Transfer a disk from tower 1 to tower 3.'.

elaborate(m21,Output) :-

    Output = 'Transfer a disk from tower 2 to tower 1.'.

elaborate(m23,Output) :-

    Output = 'Transfer a disk from tower 2 to tower 3.'.

elaborate(m31,Output) :-

    Output = 'Transfer a disk from tower 3 to tower 1.'.

elaborate(m32,Output) :-

    Output = 'Transfer a disk from tower 3 to tower 2.'.

% -----------------------------------------------------------------------

% --- Unit test programs

test\_\_m12 :-

    write('Testing: move\_m12\n'),

    TowersBefore = [[t,s,m,l,h],[],[]],

    trace('','TowersBefore',TowersBefore),

    m12(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m12x :-

    write('Testing: move\_m12\n'),

    TowersBefore = [[s,m,l,h],[],[t]],

    trace('','TowersBefore',TowersBefore),

    m12(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m13 :-

    write('Testing: move\_m13\n'),

    TowersBefore = [[t,s,m,l,h],[],[]],

    trace('','TowersBefore',TowersBefore),

    m13(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m13x :-

    write('Testing: move\_m13\n'),

    TowersBefore = [[s,m,l,h],[],[t]],

    trace('','TowersBefore',TowersBefore),

    m13(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m21 :-

    write('Testing: move\_m21\n'),

    TowersBefore = [[],[t,s,m,l,h],[]],

    trace('','TowersBefore',TowersBefore),

    m21(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m23 :-

    write('Testing: move\_m23\n'),

    TowersBefore = [[],[t,s,m,l,h],[]],

    trace('','TowersBefore',TowersBefore),

    m23(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m31 :-

    write('Testing: move\_m31\n'),

    TowersBefore = [[],[],[t,s,m,l,h]],

    trace('','TowersBefore',TowersBefore),

    m31(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_m32 :-

    write('Testing: move\_m32\n'),

    TowersBefore = [[],[],[t,s,m,l,h]],

    trace('','TowersBefore',TowersBefore),

    m32(TowersBefore,TowersAfter),

    trace('','TowersAfter',TowersAfter).

test\_\_valid\_state :-

    write('Testing: valid\_state\n'),

    test\_\_vs([[l,t,s,m,h],[],[]]),

    test\_\_vs([[t,s,m,l,h],[],[]]),

    test\_\_vs([[],[h,t,s,m],[l]]),

    test\_\_vs([[],[t,s,m,h],[l]]),

    test\_\_vs([[],[h],[l,m,s,t]]),

    test\_\_vs([[],[h],[t,s,m,l]]).

test\_\_vs(S) :-

    valid\_state(S),

    write(S), write(' is valid.'), nl.

test\_\_vs(S) :-

    write(S), write(' is invalid.'), nl.

test\_\_write\_sequence :-

    write('First test of write\_sequence ...'), nl,

    write\_sequence([m31,m12,m13,m21]),

    write('Second test of write\_sequence ...'), nl,

    write\_sequence([m13,m12,m32,m13,m21,m23,m13]).