Second Prolog Programming Assignment Specification

Learning Abstract

Tasks

1. Working within a nice text editor and with a good Prolog interpreter, do the nine enumerated tasks.
2. Craft a nicely structured document that contains representations of each of the seven tasks that you were just asked to do which produce tangible output. Moreover, be sure to title the document, and place a "learning abstract" just after the title, before presenting your work on each of the seven tasks.
3. Post your document to you web work site.

Task 1: Problem Contemplation - Towers of Hanoi

This programming challenge affords you an opportunity to implement a state space problem solver for the Towers of Hanoi problem. Please review the problem statement, and then contemplate the representation that was presented in class.

Problem Statement - Towers of Hanoi

The three peg / three tower problem: Three pegs/towers. Three disks large (L), medium (M), small (S). The disks are place on the pegs subject to the constraint that a larger disk "cannot appear" on top of a smaller disk. A move consists of transferring a disk, the top one, from one peg to another, placing it on top of whatever disks may be present. The task is to transfer all of the pegs from the first peg to the third peg.

The four peg / three tower problem: Three pegs/towers. Four disks huge (H), large (L), medium (M), small (S). The disks are place on the pegs subject to the constraint that a larger disk "cannot appear" on top of a smaller disk. A move consists of transferring a disk, the top one, from one peg to another, placing it on top of whatever disks may be present. The task is to transfer all of the pegs from the first peg to the third peg.

The five peg / three tower problem: Three pegs/towers. Five disks huge (H), large (L), medium (M), small (S), tiny (T). The disks are place on the pegs subject to the constraint that a larger disk "cannot appear" on top of a smaller disk. A move consists of transferring a disk, the top one, from one peg to another, placing it on top of whatever disks may be present. The task is to transfer all of the pegs from the first peg to the third peg.

State Space Representation - Towers of Hanoi

For the three disk problem, represent the three disks by symbols L (large) and M (medium) and S (small). Represent the three pegs as lists, imagining the disks arranged from left to right in increasing order of size.

Then ...

- I = {((S M L) () ())}
- G = {(() () (S M L))}
- O = {M12, M13, M21, M23, M31, M32}, where
  - M12 - move a disk from peg 1 to peg 2
  - M13 - move a disk from peg 1 to peg 3
  - M21 - move a disk from peg 2 to peg 1
  - M23 - move a disk from peg 2 to peg 3
  - M31 - move a disk from peg 3 to peg 1
  - M32 - move a disk from peg 3 to peg 2

One possible state space solution:

M13 ⇒ M12 ⇒ M32 ⇒ M13 ⇒ M21 ⇒ M23 ⇒ M13

For the four disk problem, represent the four disks by symbols H (huge) and L (large) and M (medium) and S (small). Represent the three pegs as lists, imagining the disks arranged from left to right in increasing order of size. Adjust the initial state and the goal state appropriately.

Then ...

- I = {((H S M L) () ())}
- G = {(() () (S M L H))}
- O = {M12, M13, M21, M23, M31, M32}, where
  - M12 - move a disk from peg 1 to peg 2
  - M13 - move a disk from peg 1 to peg 3
  - M21 - move a disk from peg 2 to peg 1
  - M23 - move a disk from peg 2 to peg 3
  - M31 - move a disk from peg 3 to peg 1
  - M32 - move a disk from peg 3 to peg 2

One possible state space solution:

M13 ⇒ M12 ⇒ M32 ⇒ M13 ⇒ M21 ⇒ M23 ⇒ M13

For the five disk problem, represent the five disks by symbols H (huge) and L (large) and M (medium) and S (small) and T (tiny). Represent the three pegs as lists, imagining the disks arranged from left to right in increasing order of size. Adjust the initial state and the goal state appropriately.

Then ...

- I = {((S M L) () ())}
- G = {(() () (S M L))}
- O = {M12, M13, M21, M23, M31, M32}, where
  - M12 - move a disk from peg 1 to peg 2
  - M13 - move a disk from peg 1 to peg 3
  - M21 - move a disk from peg 2 to peg 1
  - M23 - move a disk from peg 2 to peg 3
  - M31 - move a disk from peg 3 to peg 1
  - M32 - move a disk from peg 3 to peg 2

One possible state space solution:

M13 ⇒ M12 ⇒ M32 ⇒ M13 ⇒ M21 ⇒ M23 ⇒ M13

Task 2: Code Contemplation

Please review the Missionaries and Cannibals state space problem solving program that was presented in Lesson 7. Then, contemplate the following unrefined state space problem solving program for the Towers of Hanoi problem. In subsequent tasks, you will be asked to refine and demo this code.

When you are ready, place this text into a file called toh.pro. (Be sure **not** to place the "redacted code" tags in your file.) Also, place the inspector.pro file, provided as an appendix to this programming assignment, in your computational world as a sibling to the toh.pro file. Then, load this code into a Prolog process, just to be sure that everything is in order before you commence with the subsequent tasks.

```
% ---------------------------------------------------------------------
% -------------------------------------------------------------------% --- File: towers_of_hanoi.pro
% --- Line: Program to solve the Towers of Hanoi problem
% -------------------------------------------------------------------:- consult('inspector.pro').

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

make_move(TowersBeforeMove,TowersAfterMove,m12) :m12(TowersBeforeMove,TowersAft
   erMove).
   make_move(TowersBeforeMove,TowersAfterMove,m13) :m13(TowersBeforeMove,Towers
   AfterMove).
make_move(TowersBeforeMove,TowersAfterMove,m21) :m21(TowersBeforeMove,TowersAft
   erMove).
make_move(TowersBeforeMove,TowersAfterMove,m23) :m23(TowersBeforeMove,TowersAft
   erMove).
make_move(TowersBeforeMove,TowersAfterMove,m31) :m31(TowersBeforeMove,TowersAft
   erMove).
make_move(TowersBeforeMove,TowersAfterMove,m32) :-
   m32(TowersBeforeMove,TowersAfterMove).

<<redacted: the six state space operators>>
```

```
% -------------------------------------------------------------------% --- valid_state(S) :: S is a valid state

<<redacted: valid_state>>

% -------------------------------------------------------------------% --- solve(Start,Solution) :: succeeds if
Solution represents a path % --- from the start state to the goal state.

solve :extend_path([[[s,m,l],[],[]]],[],Solution),
   write_solution(Solution).

extend_path(PathSoFar,SolutionSoFar,Solution)  :PathSoFar  =
[[[],[],[s,m,l]]|_],                    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.

<<redacted: write_sequence>>
% -------------------------------------------------------------------% --- Unit test programs

<<redacted: the unit test programs>>
```

Task 3: One Move Predicate and a Unit Test

For this task you are given some code, and simply asked to enter it and run it. The code consists of the implementation of a state space operator, and a unit test program for the operator. You are also provided with a unit test demo.

Please note that this state space operator, as well as the other five, simply moves a disk from one peg to another, whether or not the move is "legal".

State Space Operator Implementation

Please add the following code, which implements the state space operator to move a disk from peg 1 to peg 2, m12, to your toh.pro file.

```
m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :Tower1Before = [H|T],
    Tower1After = T, Tower2Before = L,
    Tower2After = [H|L].
```

Unit Test Code

Please add the following code, which performs a unit test for the m12 predicate, to your toh.pro file.

```
test__m12 :-
    write('Testing:  move_m12\n'),  TowersBefore  =
    [[t,s,m,l,h],[],[]],
    trace('','TowersBefore',TowersBefore),
    m12(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).
```

Unit Test Demo

Please run the unit test. If it works, great! Otherwise, fix what needs to be fixed.

```
bash-3.2$ swipl <<redacted>>

?- consult('toh.pro').
% inspector.pro compiled 0.00 sec, 7 clauses % toh.pro
compiled 0.00 sec, 56 clauses true.

?- test__m12.
Testing: move_m12
TowersBefore = [[t,s,m,l,h],[],[]] TowersAfter =
[[s,m,l,h],[t],[]] true.

?-
```

Post

Please post the code that implements the state space operator, the unit test code, and the unit test demo, being sure to do so in a clear and obvious manner.

```
test__m12 :-
write('Testing: move_m12\n'),
TowersBefore = [[t,s,m,l,h],[],[]],
trace('','TowersBefore',TowersBefore),
m12(TowersBefore,TowersAfter),
trace('','TowersAfter',TowersAfter).
```

```
m12([Tower1Before,Tower2Before,_],[Tower1After,Tower2After,_]) :-
Tower1Before = [H|T],
Tower1After = T,
Tower2Before = L,
Tower2After = [H|L].
```

```
?- test__m12.
Testing: move_m12
TowersBefore' = '[[t,s,m,l,h],[],[]]
TowersAfter' = '[[s,m,l,h],[t],_7944]
true.
```

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

Please add code to implement the remaining 5 state space operators (m13 and m21 and m23 and m31 and m32). Please add a unit test program, analogous to that provided in the previous task, to test each of the five state space operators that you are asked to write for this task.

After all of the unit tests confirm that your code is good for the state space operators, perform a demo that runs all **six** unit test programs, thus assuring that all **six** state space operators are performing as they should.

Post

For this task, please post (1) the code for all **six** state space operators, (2) the code for all **six** unit test programs, and (3) the demo in which all **six** unit test programs are run.

```
m12([Tower1Before,Tower2Before,_],[Tower1After,Tower2After,_]) :-
Tower1Before = [H|T],
Tower1After = T,
Tower2Before = L,
Tower2After = [H|L].

m13([Tower1Before,_,Tower3Before],[Tower1After,_,Tower3After]) :-
Tower1Before = [H|T],
Tower1After = T,
Tower3Before = L,
Tower3After = [H|L].

m21([Tower1Before,Tower2Before,_],[Tower1After,Tower2After,_]) :-
Tower2Before = [H|T],
Tower2After = T,
Tower1Before = L,
Tower1After = [H|L].

m23([_,Tower2Before,Tower3Before],[_,Tower2After,Tower3After]) :-
Tower2Before = [H|T],
Tower2After = T,
Tower3Before = L,
Tower3After = [H|L].

m31([Tower1Before,_,Tower3Before],[Tower1After,_,Tower3After]) :-
Tower3Before = [H|T],
Tower3After = T,
Tower1Before = L,65;0;2M
Tower1After = [H|L].

m32([_,Tower2Before,Tower3Before],[_,Tower2After,Tower3After]) :-
Tower3Before = [H|T],
Tower3After = 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).

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).
```

```
?- test__m12.
Testing: move_m12
TowersBefore' = '[[t,s,m,l,h],[],[]]
TowersAfter' = '[[s,m,l,h],[t],_7944]
true.

?- test_m12.
Correct to: "test__m12"? yes
Testing: move_m12
TowersBefore' = '[[t,s,m,l,h],[],[]]
TowersAfter' = '[[s,m,l,h],[t],_9282]
true.

?- test__m12.
Testing: move_m12
TowersBefore' = '[[t,s,m,l,h],[],[]]
TowersAfter' = '[[s,m,l,h],[t],_10066]
true.

?- test__m13.
Testing: move_m13
TowersBefore' = '[[t,s,m,l,h],[],[]]
TowersAfter' = '[[s,m,l,h],_10844,[t]]
true.

?- test__m21.
Testing: move_m21
TowersBefore' = '[[],[t,s,m,l,h],[]]
TowersAfter' = '[[t],[s,m,l,h],_11634]
true.

?- test__m23.
Testing: move_m23
TowersBefore' = '[[],[t,s,m,l,h],[]]
TowersAfter' = '[_12406,[s,m,l,h],[t]]
true.

?- test__m31.
Testing: move_m31
TowersBefore' = '[[],[],[t,s,m,l,h]]
TowersAfter' = '[[t],_13196,[s,m,l,h]]
true.

?- test__m32.
Testing: move_m32
TowersBefore' = '[[],[],[t,s,m,l,h]]
TowersAfter' = '[_13974,[t],[s,m,l,h]]
true.
```

Task 5: Valid State Predicate and Unit Test

In this task you are asked to write a predicate to check whether or not a state in the Towers of Hanoi problem is valid. You are also provided with a unit test program to use in assuring that your code is sound.
Please review the given demo. Then please review the given test program, which are you required to use to assure that your predicate to check the validity of a state is sound.

Then write the predicate of one parameter called validstate to do what needs to be done, that is, to check to see that each of the three towers is properly formed. Once you have written the predicate, test it with the given unit tester. If it works, great! If not, fix your code for the validstate predicate.

Unit Test Program Demo

?- test__valid_state. Testing:
valid_state
[[l,t,s,m,h],[],[]] is invalid.
[[t,s,m,l,h],[],[]] is valid.
[[],[h,t,s,m],[l]] is invalid. [[],[t,s,m,h],[l]] is valid.
[[],[h],[l,m,s,t]] is invalid.
[[],[h],[t,s,m,l]] is valid. true

?-

Unit Test Program

```
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.
```

Post

Post (1) your code for the validstate predicate, (2) my unit test program code, and (3) your unit test program demo.

```
valid_state([P,P1,P2]) :-
        isValid(P),
        isValid(P1),
        isValid(P2).

isValid([]).
isValid([t]).
isValid([s]).
isValid([m]).
isValid([l]).
isValid([h]).
isValid([t,s]).
isValid([t,m]).
isValid([t,l]).
isValid([t,h]).
isValid([s,m]).
isValid([s,l]).
isValid([s,h]).
isValid([m,l]).
isValid([m,h]).
isValid([l,h]).
isValid([t,s,m]).
isValid([t,s,l]).
isValid([t,s,h]).
isValid([s,m,l]).
isValid([s,m,h]).
isValid([t,s,m,l,h]).
```

```
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__valid_state.
Testing: valid_state
[[l,t,s,m,h],[],[]] is invalid.
[[t,s,m,l,h],[],[]] is valid.
[[],[h,t,s,m],[l]] is invalid.
[[],[t,s,m,h],[l]] is valid.
[[],[h],[l,m,s,t]] is invalid.
[[],[h],[t,s,m,l]] is valid.
true
```

Task 6: Defining the write sequence predicate

Write the one parameter writesequence that takes a sequence of symbols corresponding to a sequence of state space operators and writes the corresponding sequence of operator descriptions, in a manner consistent with the code and demo provided.

Unit Test Program 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]).
```

Unit Test Program Demo

```
?- test__write_sequence.
First test of write_sequence ...
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
```

Transfer a disk from tower 2 to tower 1. Second test of write_sequence ...
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 2 to tower 3.
Transfer a disk from tower 1 to tower 3. true.

?-

Post

Post (1) your code for the writesequence predicate, (2) my unit tester code, and (3) your unit test program demo.

```
write_sequence([]).
write_sequence([H|T]) :-
        printMove(H),
        write_sequence(T).

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]).
```

```
?- test__write_sequence.
First test of write_sequence ...
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 1
Second test of write_sequence ...
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 1 to tower 3
true
```

Task 7: Run the program to solve the 3 disk problem

Run the program to solve the three disk Towers of Hanoi problem:

1. With the intetermediate output displayed.
2. With just the English-like solution displayed.

Do your best to answer the following questions:

1. What was the length of your program's solution to the three disk problem?
2. What is the length of the shortest solution to the three disk problem?
3. How do you account for the discrepency?

What to Post?

```
?- solve.
PathSoFar' = '[[[s,m,l],[],[]]]
Move' = 'm12
NextState' = '[[m,l],[s],_8954]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]]]
Move' = 'm12
NextState' = '[[l],[m,s],_9208]
Move' = 'm13
NextState' = '[[l],_9202,[m]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]]]
Move' = 'm12
NextState' = '[[],[l],_9310]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]]]
Move' = 'm21
NextState' = '[[l],[],_9424]
Move' = 'm23
NextState' = '[_9412,[],[l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]]]
Move' = 'm31
NextState' = '[[l],_9544,[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[],[]]]
Move' = 'm12
NextState' = '[[],[l],_9688]
Move' = 'm13
NextState' = '[[],_9682,[l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]]]
Move' = 'm12
NextState' = '[[],[l,t],_9694]
Move' = 'm13
NextState' = '[[],_9688,[l]]
Move' = 'm21
NextState' = '[[t,l],[],_9694]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]]]
Move' = 'm12
NextState' = '[[l],[t],_9844]
Move' = 'm13
NextState' = '[[l],_9838,[t]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[],[t]]]
Move' = 'm12
NextState' = '[[],[l],_10006]
Move' = 'm13
NextState' = '[[],_10000,[l,t]]
Move' = 'm31
NextState' = '[[t,l],_10000,[]]
Move' = 'm32
NextState' = '[_9994,[t],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[t],[t]]]
Move' = 'm12
NextState' = '[[],[l,t],_10012]
Move' = 'm13
NextState' = '[[],_10006,[l,t]]
Move' = 'm21
NextState' = '[[t,l],[],_10012]
Move' = 'm23
NextState' = '[_10000,[],[t,t]]
Move' = 'm31
NextState' = '[[t,l],_10006,[]]
Move' = 'm32
NextState' = '[_10000,[t,t],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]]]
Move' = 'm12
NextState' = '[[],[l,s],_10012]
Move' = 'm13
NextState' = '[[],_10006,[l,t]]
[0] 0:[tmux]*Z
```

```
Move' = 'm21
NextState' = '[[s,l],[],_10012]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]]]
Move' = 'm12
NextState' = '[[l],[s],_10186]
Move' = 'm13
NextState' = '[[l],_10180,[s]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[],[s]]]
Move' = 'm12
NextState' = '[[],[l],_10372]
Move' = 'm13
NextState' = '[[],_10366,[l,s]]
Move' = 'm31
NextState' = '[[s,l],_10366,[]]
Move' = 'm32
NextState' = '[_10360,[s],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]]]
Move' = 'm12
NextState' = '[[],[l,t],_10378]
Move' = 'm13
NextState' = '[[],_10372,[l,s]]
Move' = 'm21
NextState' = '[[t,l],[],_10378]
Move' = 'm23
NextState' = '[_10366,[],[t,s]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]]]
Move' = 'm31
NextState' = '[[t],_10570,[s]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]]]
Move' = 'm12
NextState' = '[[],[t],_10786]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]]]
Move' = 'm21
NextState' = '[[t],[],_11008]
Move' = 'm23
NextState' = '[_10996,[],[t]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]]]
Move' = 'm31
NextState' = '[[t],_11236,[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[],[]]]
Move' = 'm12
NextState' = '[[],[t],_11488]
Move' = 'm13
NextState' = '[[],_11482,[t]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[t],[]]]
Move' = 'm12
NextState' = '[[],[t,t],_11494]
Move' = 'm13
NextState' = '[[],_11488,[t]]
Move' = 'm21
NextState' = '[[t,t],[],_11494]
Move' = 'm23
NextState' = '[_11482,[],[t]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]]]
Move' = 'm12
NextState' = '[[],[t,s],_11494]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[]]]
Move' = 'm21
NextState' = '[[t],[s],_11752]
Move' = 'm23
NextState' = '[_11740,[s],[t]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[t]]]
Move' = 'm21
NextState' = '[[t],[s],_11758]
Move' = 'm23
[0] 0:[tmux]*Z                                                                    "altair" 00:29 25-Nov
```

```
NextState' = '[_11740,[s],[t]]                                                                                                    [298/1926]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[t]]]
Move' = 'm21
NextState' = '[[t],[s],_11758]
Move' = 'm23
NextState' = '[_11746,[s],[t,t]]
Move' = 'm31
NextState' = '[[t],_11752,[]]
Move' = 'm32
NextState' = '[_11746,[t,t,s],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]]]
Move' = 'm21
NextState' = '[[t],[s],_11758]
Move' = 'm23
NextState' = '[_11746,[s],[t,s]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]]]
Move' = 'm21
NextState' = '[[s],[],_12028]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[]]]
Move' = 'm12
NextState' = '[[],[s],_12310]
Move' = 'm13
NextState' = '[[],_12304,[s]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]]]
Move' = 'm12
NextState' = '[[],[s],_12316]
Move' = 'm13
NextState' = '[[],_12310,[s,t]]
Move' = 'm31
NextState' = '[[t,s],_12310,[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]]]
Move' = 'm12
NextState' = '[[s],[t],_12610]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[]]]
Move' = 'm12
NextState' = '[[],[s,t],_12916]
Move' = 'm13
NextState' = '[[],_12910,[s]]
Move' = 'm21
NextState' = '[[t,s],[],_12916]
Move' = 'm23
NextState' = '[_12904,[],[t]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[t]]]
Move' = 'm12
NextState' = '[[],[s,t],_12922]
Move' = 'm13
NextState' = '[[],_12916,[s,t]]
Move' = 'm21
NextState' = '[[t,s],[],_12922]
Move' = 'm23
NextState' = '[_12910,[],[t,t]]
Move' = 'm31
NextState' = '[[t,s],_12916,[]]
Move' = 'm32
NextState' = '[_12910,[t,t],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[s]]]
Move' = 'm12
NextState' = '[[],[s,t],_12922]
[0] 0:[tmux]*Z                                                                    "altair" 00:30 25-Nov-21
```

```
Move' = 'm13                                                                                                                              [233/1926]
NextState' = '[[],_12916,[s,s]]
Move' = 'm21
NextState' = '[[t,s],[],_12922]
Move' = 'm23
NextState' = '[_12910,[],[t,s]]
Move' = 'm31
NextState' = '[[s,s],_12916,[]]
Move' = 'm32
NextState' = '[_12910,[s,t],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]]]
Move' = 'm12
NextState' = '[[],[s,t],_12922]
Move' = 'm13
NextState' = '[[],_12916,[s,m]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]]]
Move' = 'm31
NextState' = '[[s],_13234,[m]]
Move' = 'm32
NextState' = '[_13228,[s],[m]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]]]
Move' = 'm21
NextState' = '[[s],[],_13570]
Move' = 'm23
NextState' = '[_13558,[],[s,m]]
Move' = 'm31
NextState' = '[[m],_13564,[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]]]
Move' = 'm12
NextState' = '[[],[m],_13912]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[]]]
Move' = 'm21
NextState' = '[[m],[],_14266]
Move' = 'm23
NextState' = '[_14254,[],[m]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]]]
Move' = 'm21
NextState' = '[[m],[],_14272]
Move' = 'm23
NextState' = '[_14260,[],[m,t]]
Move' = 'm31
NextState' = '[[t],_14266,[]]
Move' = 'm32
NextState' = '[_14260,[t,m],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]]]
Move' = 'm21
NextState' = '[[t],[m],_14638]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[]]]
Move' = 'm12
NextState' = '[[],[t,m],_15016]
Move' = 'm13
NextState' = '[[],_15010,[t]]
Move' = 'm21
NextState' = '[[m,t],[],_15016]
Move' = 'm23
NextState' = '[_15004,[],[m]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
```

```
NextState' = '[[m,s],[],_15820]                                                                                                        [106/1926
Move' = 'm23
NextState' = '[_15808,[],[m,s]]
Move' = 'm31
NextState' = '[[s,s],_15814,[]]
Move' = 'm32
NextState' = '[_15808,[s,m],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[m]]]
Move' = 'm12
NextState' = '[[],[s,m],_15820]
Move' = 'm13
NextState' = '[[],_15814,[s,m]]
Move' = 'm21
NextState' = '[[m,s],[],_15820]
Move' = 'm23
NextState' = '[_15808,[],[m,m]]
Move' = 'm31
NextState' = '[[m,s],_15814,[]]
Move' = 'm32
NextState' = '[_15808,[m,m],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]]]
Move' = 'm12
NextState' = '[[],[s,m],_15820]
Move' = 'm13
NextState' = '[[],_15814,[s,l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]]]
Move' = 'm31
NextState' = '[[s],_16228,[l]]
Move' = 'm32
NextState' = '[_16222,[s],[l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[],[s],[l]]]
Move' = 'm21
NextState' = '[[s],[],_16660]
Move' = 'm23
NextState' = '[_16648,[],[s,l]]
Move' = 'm31
NextState' = '[[l],_16654,[]]
Move' = 'm32
NextState' = '[_16648,[l,s],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]]]
Move' = 'm12
NextState' = '[[],[t,s],_16666]
Move' = 'm13
NextState' = '[[],_16660,[t,l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]]]
Move' = 'm31
NextState' = '[[t],_17098,[l]]
Move' = 'm32
NextState' = '[_17092,[t],[l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]],[[],[t],[l]]]
Move' = 'm21
NextState' = '[[t],[],_17554]
Move' = 'm23
NextState' = '[_17542,[],[t,l]]
Move' = 'm31
NextState' = '[[l],_17548,[]]
Move' = 'm32
NextState' = '[_17542,[l,t],[]]
[0] 0:[tmux]*Z                                                                                                          "altair" 00:33 25 Nov
```

```
NextState' = '[[l],_17548,[]]                                                                    [44/1926]
Move' = 'm32
NextState' = '[_17542,[l,t],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]],[[t],[t],[l]]]
Move' = 'm12
NextState' = '[[],[t,t],_17560]
Move' = 'm13
NextState' = '[[],_17554,[t,l]]
Move' = 'm21
NextState' = '[[t,t],[],_17560]
Move' = 'm23
NextState' = '[_17548,[],[t,l]]
Move' = 'm31
NextState' = '[[l,t],_17554,[]]
Move' = 'm32
NextState' = '[_17548,[l,t],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]],[[s],[t],[l]]]
Move' = 'm12
NextState' = '[[],[s,t],_17560]
Move' = 'm13
NextState' = '[[],_17554,[s,l]]
Move' = 'm21
NextState' = '[[t,s],[],_17560]
Move' = 'm23
NextState' = '[_17548,[],[t,l]]
Move' = 'm31
NextState' = '[[l,s],_17554,[]]
Move' = 'm32
NextState' = '[_17548,[l,t],[]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]],[[m],[t],[l]]]
Move' = 'm12
NextState' = '[[],[m,t],_17560]
Move' = 'm13
NextState' = '[[],_17554,[m,l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]],[[m],[t],[l]],[[],[],[m,l]]]
Move' = 'm31
NextState' = '[[m],_18016,[l]]
Move' = 'm32
NextState' = '[_18010,[m],[l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]],[[m],[t],[l]],[[],[t],[m,l]]]
Move' = 'm21
NextState' = '[[t],[],_18028]
Move' = 'm23
NextState' = '[_18016,[],[t,m,l]]
Move' = 'm31
NextState' = '[[m],_18022,[l]]
Move' = 'm32
NextState' = '[_18016,[m,t],[l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]],[[m],[t],[l]],[[],[s],[m,l]]]
Move' = 'm21
NextState' = '[[s],[],_18028]
Move' = 'm23
NextState' = '[_18016,[],[s,m,l]]
PathSoFar' = '[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[],[m]],[[],[l],[]],[[],[],[l]],[[l],[t],[]],[[t,l],[],[]],[[l],[s],[t]],[[s,l],[],[]],[[l],[t],[s]],[[],[],[t,s]],[[t],[],[s]],[[],[t],[]],[[],[],[t]],[[t],[s],[]],[[],[t,s],[s]],[[],[s
],[t,s]],[[s],[],[t]],[[t,s],[],[]],[[s],[t],[m]],[[],[],[s,m]],[[],[s],[m]],[[m],[],[]],[[],[m],[t]],[[],[t,m],[]],[[t],[m],[s]],[[],[s,m],[]],[[s],[m],[l]],[[],[],[s,l]],[[t],[s],[l]],[[],[],[t,l]],[[m],[t],[l]],[[],[s],[m,l]],[[],[],[
s,m,l]]]
SolutionSoFar' = '[m12,m13,m12,m23,m31,m21,m13,m21,m13,m23,m31,m12,m23,m31,m12,m23,m21,m31,m12,m13,m32,m31,m12,m32,m21,m32,m21,m13,m32,m13,m32,m13,m23]
```

```
Solution ...

Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 3
```

1. What was the length of your program's solution to the three disk problem?

   33 moves

2. What is the length of the shortest solution to the three disk problem?

   7

3. How do you account for the discrepancy?

   I didn't have my code remove unnecessary intermediate moves.

Task 8: Run the program to solve the 4 disk problem

Run the program to solve the four disk Towers of Hanoi problem, without displaying any intermediate output. Convince yourself that your program does, indeed, find a solution. If you can't do so, fix your program.

Do your best to answer the following questions:

1. What was the length of your program's solution to the four disk problem?
2. What is the length of the shortest solution to the four disk problem?

Post

There is too many lines of out put to post the intermediate step.

```
Solution ...

Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 1
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 1 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 3
Transfer a disk from tower 3 to tower 1
Transfer a disk from tower 3 to tower 2
Transfer a disk from tower 1 to tower 3
Transfer a disk from tower 2 to tower 3

true .

[0] 0:[tmux]*Z
```

Task 9: Review your code and archive it

Review your program to make sure that it is properly formatted. Fix it up, if need be, Then be sure to run your program to make sure that the code is still sound.

What to Post?

```
% ---------------------------------------------------------------------------
% ---------------------------------------------------------------------------
% --- File: towers_of_hanoi.pro
% --- Line: Program to solve the Towers of Hanoi problem
% ---------------------------------------------------------------------------
:- consult('inspector.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,_],[Tower1After,Tower2After,_]) :-
Tower1Before = [H|T],
Tower1After = T,
Tower2Before = L,
Tower2After = [H|L].

m13([Tower1Before,_,Tower3Before],[Tower1After,_,Tower3After]) :-
Tower1Before = [H|T],
Tower1After = T,
Tower3Before = L,
Tower3After = [H|L].

m21([Tower1Before,Tower2Before,_],[Tower1After,Tower2After,_]) :-
Tower2Before = [H|T],
Tower2After = T,
Tower1Before = L,
Tower1After = [H|L].

m23([_,Tower2Before,Tower3Before],[_,Tower2After,Tower3After]) :-
Tower2Before = [H|T],
Tower2After = T,
Tower3Before = L,
Tower3After = [H|L].

m31([Tower1Before,_,Tower3Before],[Tower1After,_,Tower3After]) :-
Tower3Before = [H|T],
Tower3After = T,
Tower1Before = L,
Tower1After = [H|L].

m32([_,Tower2Before,Tower3Before],[_,Tower2After,Tower3After]) :-
Tower3Before = [H|T],
Tower3After = T,
Tower2Before = L,
Tower2After = [H|L].

% ----------------------------------------------------------------------------
% --- valid_state(S) :: S is a valid state

size(t,0).
size(s,1).
size(m,2).
size(l,3).
size(h,4).

nth(0,[H|_], H).
nth(N, [_|T], NT) :- K is N - 1, nth(K, T, NT).

rest([_|R],R).

valid_state([P,P1,P2]) :-
        isValid(P),
        isValid(P1),
        isValid(P2).

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

%This made me run out of space on the stack.
%checkValid([]).
%checkValid(S) :-
%       %write(S),nl,
%       nth(0, S, Z),
%       %write(Z),nl,
%       nth(1, S, M),
%       %write(M),nl,
%       size(Z,N),
%       %write(N),nl,
%       size(M,P),
%       %write(P),nl,
%       %write(N), write('<'),write(P),nl,
%       N < P,
%       rest(S,R),
```

```
%       rest(S,R),
%       %write(R),nl,
%       checkValid(R).
%
%checkValid(S) :-
%       rest(S,R),
%       R = [].

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.

% ------------------------------------------------------------------------------
% --- solve(Start,Solution) :: succeeds if Solution represents a path
% --- from the start state to the goal state.
solve :-
extend_path([[[t,s,m,l],[],[]]],[],Solution),
write_solution(Solution).

extend_path(PathSoFar,SolutionSoFar,Solution) :-
PathSoFar = [[[],[],[s,m,l]]|_],
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.

printMove(H) :-
        H == m12,
        write('Transfer a disk from tower 1 to tower 2'),nl.
printMove(H) :-
        H == m13,
        write('Transfer a disk from tower 1 to tower 3'),nl.
printMove(H) :-
        H == m21,
        write('Transfer a disk from tower 2 to tower 1'),nl.
printMove(H) :-
        H == m23,
        write('Transfer a disk from tower 2 to tower 3'),nl.
printMove(H) :-
        H == m31,
        write('Transfer a disk from tower 3 to tower 1'),nl.
printMove(H) :-
        H == m32,
        write('Transfer a disk from tower 3 to tower 2'),nl.


write_sequence([]).
write_sequence([H|T]) :-
        printMove(H),
        write_sequence(T).

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]).

% ------------------------------------------------------------------------------
% --- Unit test programs
% <<redacted: the 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),
```

```
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).

```