Second Prolog Programming Assignment

Abstract

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).
```

```
?- consult('C:/Users/User/Documents/towers_of_hanoi.pro').
true.

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

Task 4: Valid State Predicate and Unit Test

```
% ------------------------------------------------------------------------
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).
```

```
?- consult('C:/Users/User/Documents/towers_of_hanoi.pro').
true.

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

?- test__m13.
Testing: move_m13
TowersBefore = [[t,s,m,l,h],[],[]]
TowersAfter = [[s,m,l,h],[],[t]]
true.

?- test__m21.
Testing: move_m21
TowersBefore = [[t,s,m,l,h],[],[]]
false.

?- test__m23.
Testing: move_m23
TowersBefore = [[t,s,m,l,h],[],[]]
false.

?- test__m31.
Testing: move_m31
TowersBefore = [[t,s,m,l,h],[],[]]
false.

?- test__m32.
Testing: move_m32
TowersBefore = [[t,s,m,l,h],[],[]]
false.
```

Task 5: Valid State Predicate and Unit Test

```
% ------------------------------------------------------------------------
% --- valid_state(S) :: S is a valid state
valid_state([Peg_1,Peg_2,Peg_3]) :-
valid_peg(Peg_1),valid_peg(Peg_2),valid_peg(Peg_3).

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

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

valid_peg([m]).
valid_peg([m,l]).
valid_peg([m,h]).
valid_peg([m,l,h]).

valid_peg([l]).
valid_peg([l,h]).

valid_peg([h]).
valid_peg([]).
```

```
?- consult('C:/Users/User/Documents/towers_of_hanoi.pro').
true.

?- 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

```
124
125 write_solution(S) :-
126 nl, write('Solution ...'), nl, nl,
127 reverse(S,R),
128 write_sequence(R),nl.
129
130 write_sequence([]).
131
132 write_sequence([H|T]) :-
133 elaborate(H,E),
134 write(E),nl,
135 write_sequence(T).
136
137 elaborate(m12,Elaboration) :-
138 Elaboration='Transfer A Disk From Tower 1 to Tower 2.'.
139
140 elaborate(m13,Elaboration) :-
141 Elaboration='Transfer A Disk From Tower 1 to Tower 3.'.
142
143 elaborate(m21,Elaboration) :-
144 Elaboration='Transfer A Disk From Tower 2 to Tower 1.'.
145
146 elaborate(m23,Elaboration) :-
147 Elaboration='Transfer A Disk From Tower 2 to Tower 3.'.
148
149 elaborate(m31,Elaboration) :-
150 Elaboration='Transfer A Disk From Tower 3 to Tower 1.'.
151
152 elaborate(m32,Elaboration) :-
153 Elaboration='Transfer A Disk From Tower 3 to Tower 2.'.
154
```

```
?- consult('C:/Users/User/Documents/towers_of_hanoi.pro').
true.

?- 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

```
?- consult('C:/Users/User/Documents/towers_of_hanoi.pro').
true.

?- solve.
PathSoFar = [[[s,m,l],[],[]]]
Move = m12
NextState = [[m,l],[s],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]]]
Move = m12
NextState = [[l],[m,s],[]]
Move = m13
NextState = [[l],[s],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]]]
Move = m12
NextState = [[],[l,s],[m]]
Move = m13
NextState = [[],[s],[l,m]]
Move = m21
NextState = [[s,l],[],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]]]
Move = m12
NextState = [[l],[s],[m]]
Move = m13
NextState = [[l],[],[s,m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]]]
Move = m12
NextState = [[],[l],[s,m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]]]
Move = m21
NextState = [[l],[],[s,m]]
Move = m23
NextState = [[],[],[l,s,m]]
Move = m31
NextState = [[s],[l],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]]]
Move = m12
NextState = [[],[s,l],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]]]
Move = m21
NextState = [[s],[l],[m]]
Move = m23
NextState = [[],[l],[s,m]]
Move = m31
NextState = [[m],[s,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]]]
Move = m12
```

```
Move = m12
NextState = [[],[m,s,l],[]]
Move = m13
NextState = [[],[s,l],[m]]
Move = m21
NextState = [[s,m],[l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
]]
Move = m12
NextState = [[m],[s,l],[]]
Move = m13
NextState = [[m],[l],[s]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[m],[l],[s]]]
Move = m12
NextState = [[],[m,l],[s]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[m],[l],[s]],[[],[m,l],[s]]]
Move = m21
NextState = [[m],[l],[s]]
Move = m23
NextState = [[],[l],[m,s]]
Move = m31
NextState = [[s],[m,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[m],[l],[s]],[[],[m,l],[s]],[[s],[m,l],[]]]
Move = m12
NextState = [[],[s,m,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[m],[l],[s]],[[],[m,l],[s]],[[s],[m,l],[]],[[],[s,m,l],[]]]
Move = m21
NextState = [[s],[m,l],[]]
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[m,l],[s]]
Move = m21
NextState = [[m,s],[l],[]]
Move = m23
NextState = [[s],[l],[m]]
Move = m32
NextState = [[],[s,m,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[m],[l],[s]],[[],[m,l],[s]],[[],[s,m,l],[]]]
Move = m21
```

```
Move = m21
NextState = [[s],[m,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[m],[l],[s]],[[],[m,l],[s]],[[],[s,m,l],[]],[[s],[m,l],[]]]
Move = m12
NextState = [[],[s,m,l],[]]
Move = m13
NextState = [[],[m,l],[s]]
Move = m21
NextState = [[m,s],[l],[]]
Move = m23
NextState = [[s],[l],[m]]
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[l],[m,s]]
Move = m21
NextState = [[l,m],[],[s]]
Move = m23
NextState = [[m],[],[l,s]]
Move = m31
NextState = [[s,m],[l],[]]
Move = m32
NextState = [[m],[s,l],[]]
Move = m21
NextState = [[l,s,m],[],[]]
Move = m23
NextState = [[s,m],[],[l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[s,m],[],[l]]]
Move = m12
NextState = [[m],[s],[l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[s,m],[],[l]],[[m],[s],[l]]]
Move = m12
NextState = [[],[m,s],[l]]
Move = m13
NextState = [[],[s],[m,l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]]]
Move = m21
NextState = [[s],[],[m,l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]],[[s],[],[m,l]]]
Move = m12
NextState = [[],[s],[m,l]]
Move = m12
NextState = [[],[s],[m,l]]
Move = m13
NextState = [[],[],[s,m,l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[]
,[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]
],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]],[[s],[],[m,l]],[[],[],[s,m,l]]]
SolutionSoFar = [m12,m13,m21,m13,m12,m31,m12,m31,m21,m23,m12,m13,m21,m13]
```

```
Solution ...

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.
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 1.
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 1 to Tower 3.
Transfer A Disk From Tower 2 to Tower 1.
Transfer A Disk From Tower 1 to Tower 3.

true
```

1. 14 moves
2. 7 moves
3. The program was constructed to test all possible paths, checking the validity of all moves and then making the corresponding move that the program traced thru.

Task 8: Run the Program to Solve the 4 Disk Problem

```
Solution ...

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 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 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 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.
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 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 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 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.
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.
```

1. 35 moves
2. 15 moves

Task 9: Review Your Code and Archive It

```
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- File: towers_of_hanoi.pro
% --- Line: Program to solve the Towers of Hanoi problem
% ------------------------------------------------------------------------
:- consult('C:/Users/User/Documents/                    ro').
% ------------------------------------------------------------------------
% --- 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([Peg_1,Peg_2,Peg_3]) :-
valid_peg(Peg_1),valid_peg(Peg_2),valid_peg(Peg_3).

valid_peg([t]).
valid_peg([t,s]).
valid_peg([t,m]).
valid_peg([t,l]).
```

```
% ------------------------------------------------------
% --- valid_state(S) :: S is a valid state
valid_state([Peg_1,Peg_2,Peg_3]) :-
valid_peg(Peg_1),valid_peg(Peg_2),valid_peg(Peg_3).

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

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

valid_peg([m]).
valid_peg([m,l]).
valid_peg([m,h]).
valid_peg([m,l,h]).

valid_peg([l]).
valid_peg([l,h]).

valid_peg([h]).
valid_peg([]).
```

```
% ------------------------------------------------------------------------
% --- 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.

write_sequence([]).

write_sequence([H|T]) :-
elaborate(H,E),
write(E),nl,
```

```
write_sequence([H|T]) :-
elaborate(H,E),
write(E),nl,
write_sequence(T).

elaborate(m12,Elaboration) :-
Elaboration='Transfer A Disk From Tower 1 to Tower 2.'.

elaborate(m13,Elaboration) :-
Elaboration='Transfer A Disk From Tower 1 to Tower 3.'.

elaborate(m21,Elaboration) :-
Elaboration='Transfer A Disk From Tower 2 to Tower 1.'.

elaborate(m23,Elaboration) :-
Elaboration='Transfer A Disk From Tower 2 to Tower 3.'.

elaborate(m31,Elaboration) :-
Elaboration='Transfer A Disk From Tower 3 to Tower 1.'.

elaborate(m32,Elaboration) :-
Elaboration='Transfer A Disk From Tower 3 to Tower 2.'.

% ----------------------------------------------------------------------
% ---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) :-
```

```
test__vs(S) :-
valid_state(S),
write(S), write(' is valid.'), nl.
test__vs(S) :-
write(S), write(' is invalid.'), nl.

% ----------------------------------------------------------------------
% --- 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).

% ----------------------------------------------------------------------
% --- Unit Test Program

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]).
```