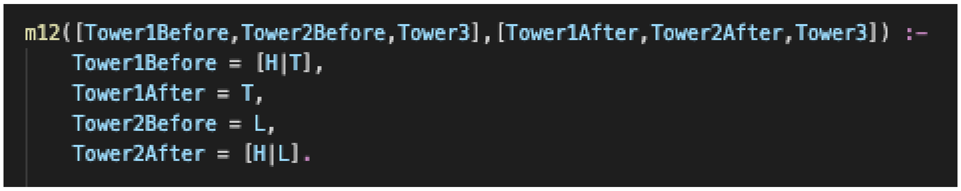
Second Prolog Programming Assignment

Learning Abstract:

In this assignment I learned how to implement a state space problem solver to solve the three disk and four disk Tower of Hanoi problem in Prolog. This involved writing state space operators to move disks, a predicate to test the validity of disks on pegs, a predicate that translated the moves into english, along with tester code to see what was happening in the world.

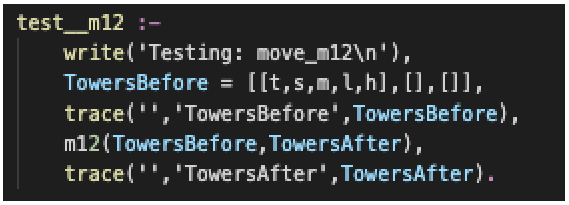
Task 3 - One Move Predicate and a Unit Test

State space operator code:



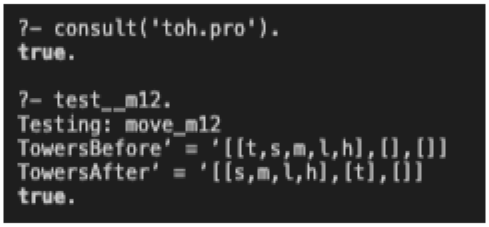
```
m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
    Tower1Before = [H|T],
    Tower1After = T,
    Tower2Before = L,
    Tower2After = [H|L].
```

Tester code:



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

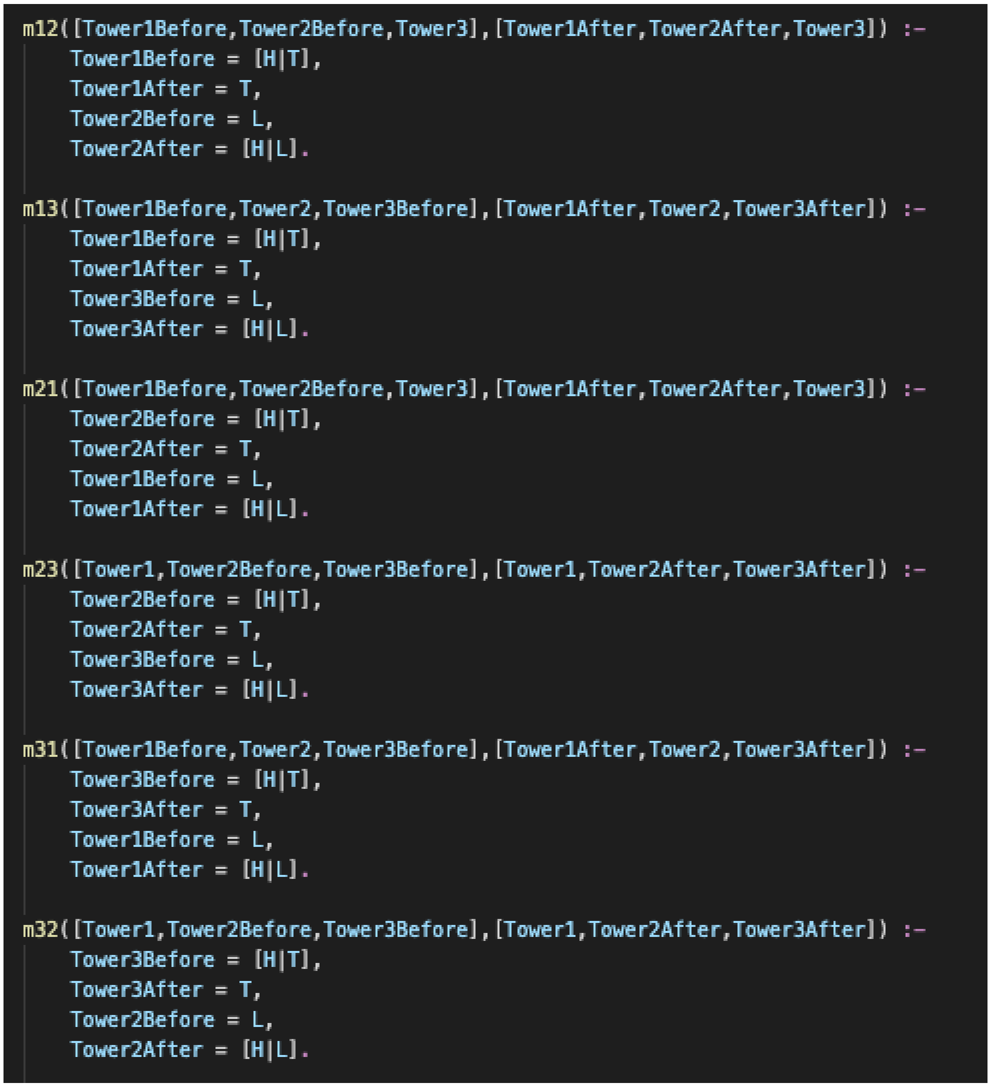
Demo:



```
?- consult('toh.pro').
true.

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

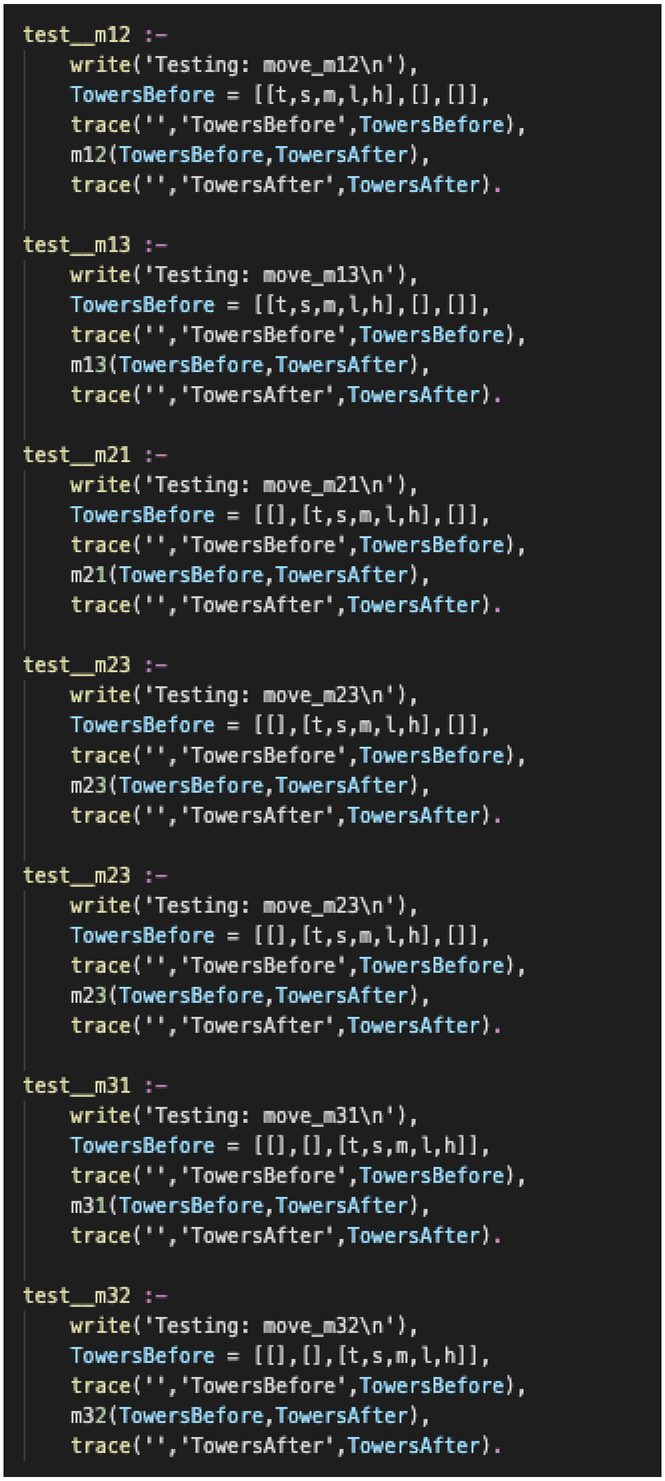
Task 4 - The Remaining Five Move Predicates and a Unit Tests

State space operators code:



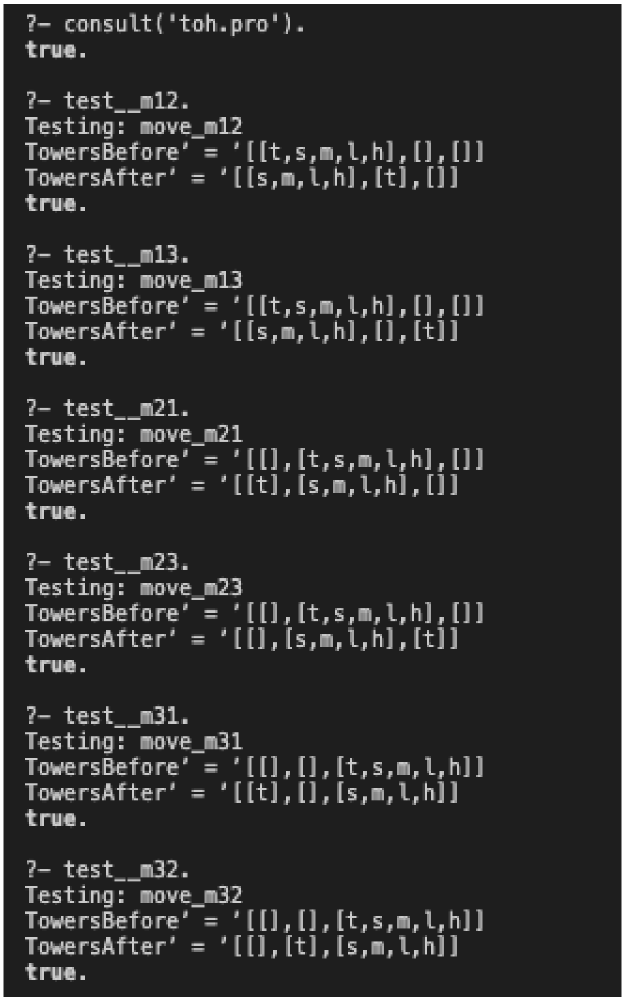
```
m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
    Tower1Before = [H|T],
    Tower1After = T,
    Tower2Before = L,
    Tower2After = [H|L].

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].
```

Tester code:



```
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__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).
```

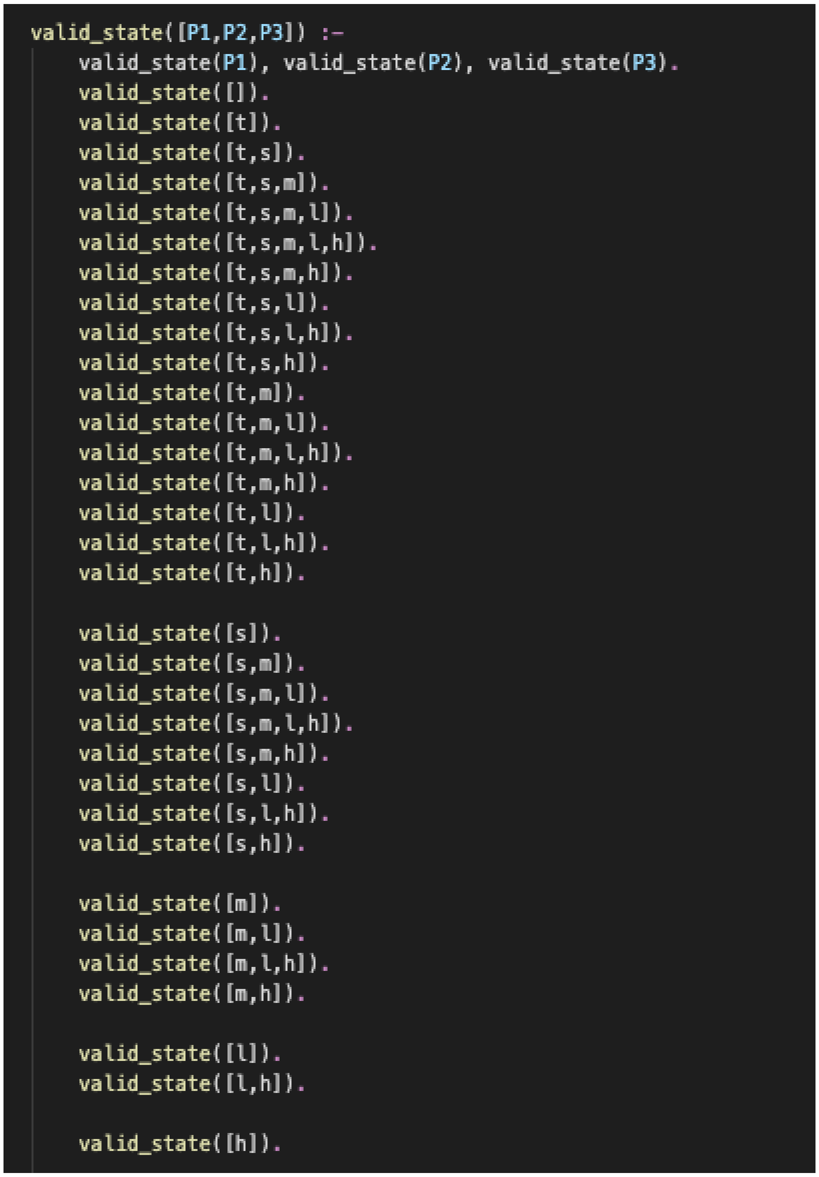
Demo:



```
?- consult('toh.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],[]]
TowersAfter’ = ‘[[t],[s,m,l,h],[]]
true.

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

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

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

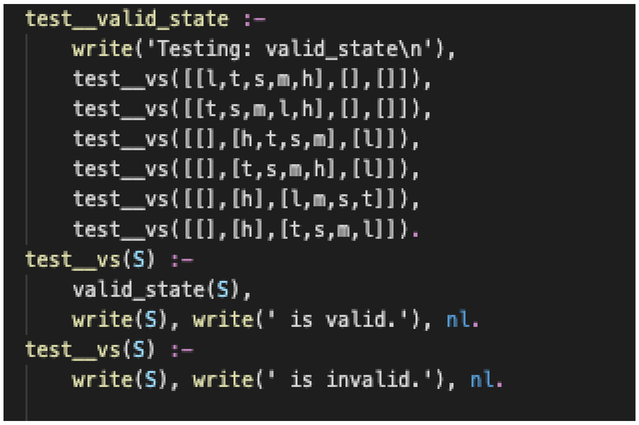
Task 5 - Valid State Predicate and Unit Test

Predicate code:



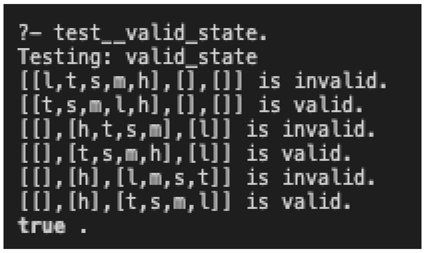
```
valid_state([P1,P2,P3]) :-
    valid_state(P1), valid_state(P2), valid_state(P3).
    valid_state([]).
    valid_state([t]).
    valid_state([t,s]).
    valid_state([t,s,m]).
    valid_state([t,s,m,l]).
    valid_state([t,s,m,l,h]).
    valid_state([t,s,m,h]).
    valid_state([t,s,l]).
    valid_state([t,s,l,h]).
    valid_state([t,s,h]).
    valid_state([t,m]).
    valid_state([t,m,l]).
    valid_state([t,m,l,h]).
    valid_state([t,m,h]).
    valid_state([t,l]).
    valid_state([t,l,h]).
    valid_state([t,h]).

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

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

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

    valid_state([h]).
```

Tester 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.
```

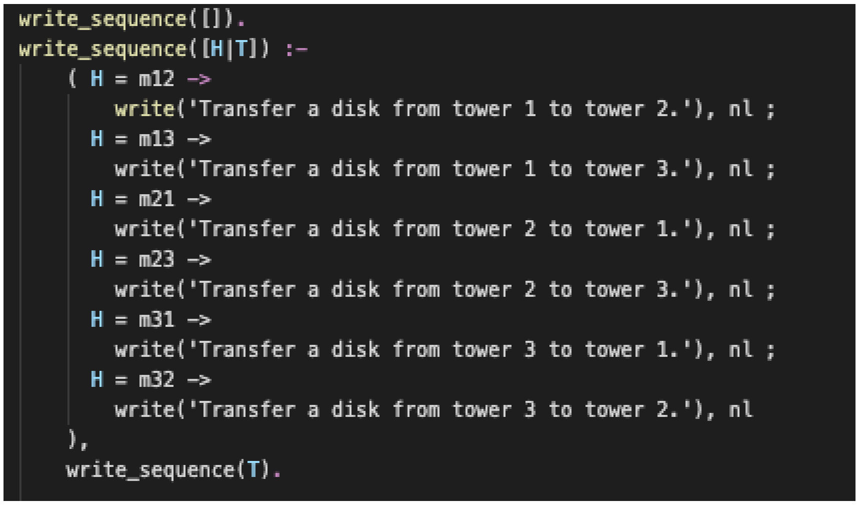
Demo:



```
?- 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 .
```

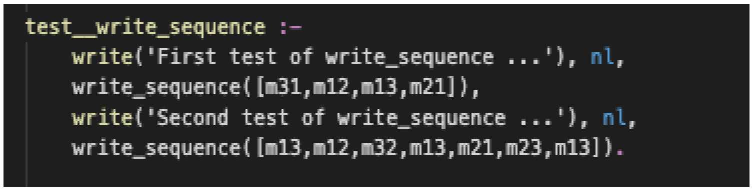
Task 6: Defining the Write Sequence Predicate

Predicate code:



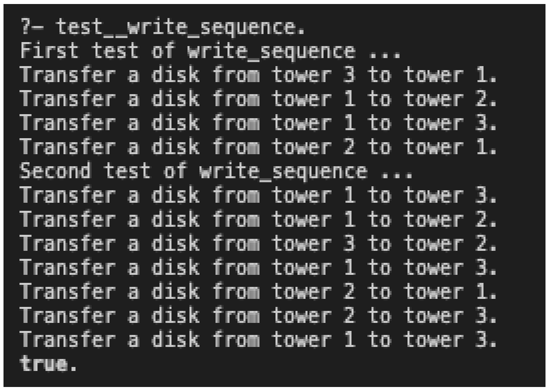
```
write_sequence([]).
write_sequence([H|T]) :-
    ( H = m12 ->
        write('Transfer a disk from tower 1 to tower 2.'), nl ;
      H = m13 ->
        write('Transfer a disk from tower 1 to tower 3.'), nl ;
      H = m21 ->
        write('Transfer a disk from tower 2 to tower 1.'), nl ;
      H = m23 ->
        write('Transfer a disk from tower 2 to tower 3.'), nl ;
      H = m31 ->
        write('Transfer a disk from tower 3 to tower 1.'), nl ;
      H = m32 ->
        write('Transfer a disk from tower 3 to tower 2.'), nl
    ),
    write_sequence(T).
```

Tester 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]).
```

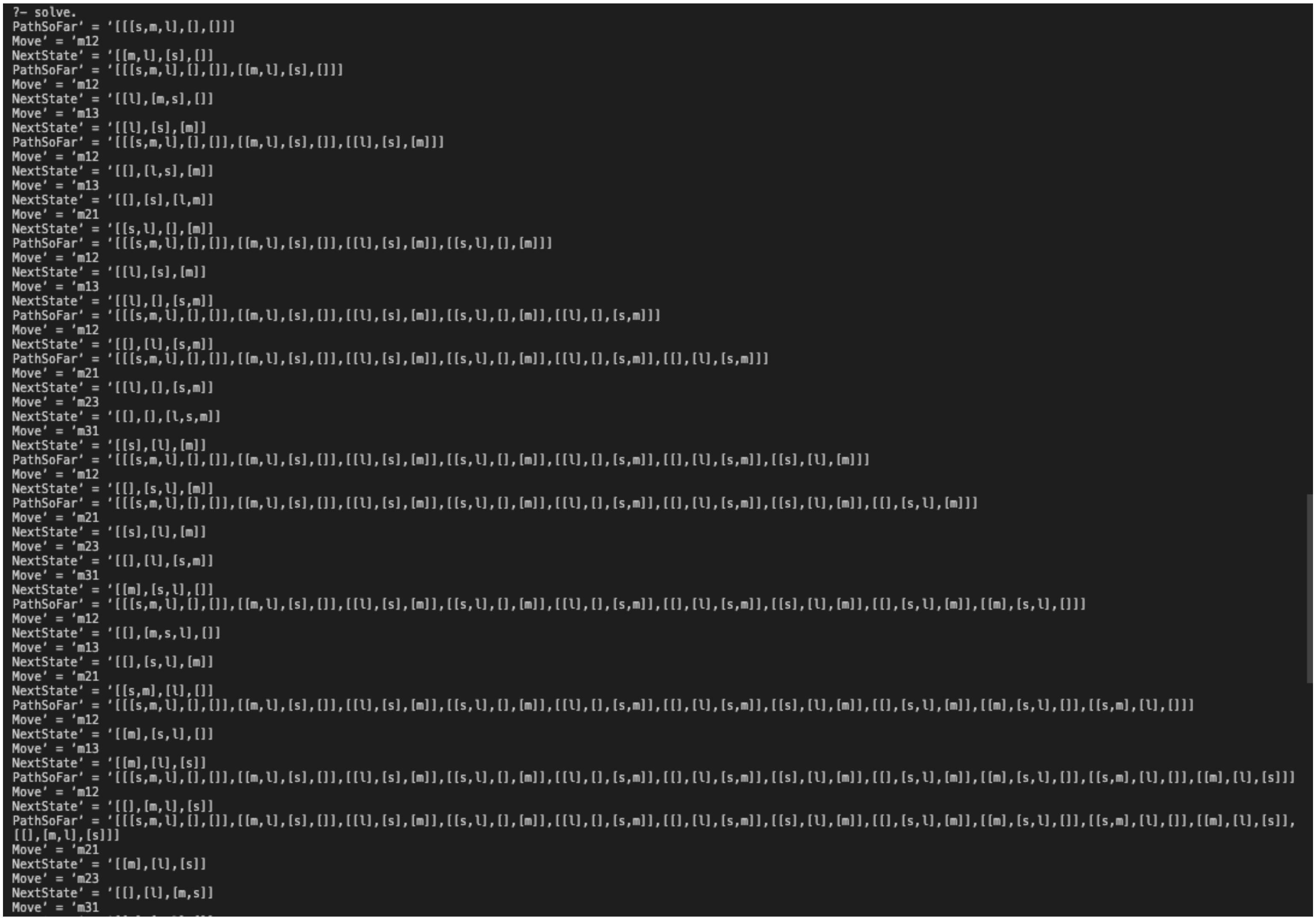
Demo:



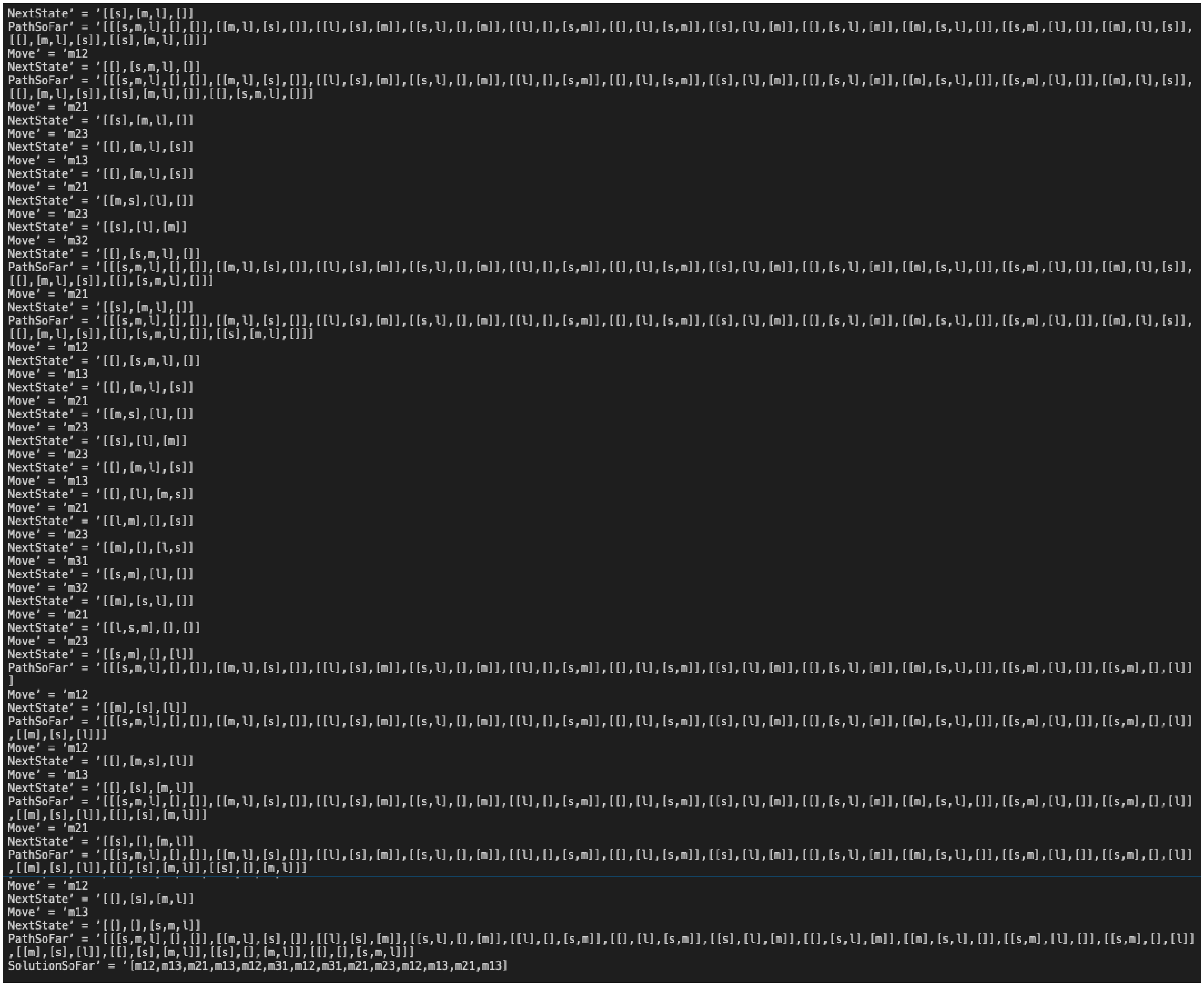
```
?- test__write_sequence.
First test of write_sequence ...
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Second test of write_sequence ...
Transfer a disk from tower 1 to tower 3.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

Task 7: Run the Program to Solve the 3 Disk Problem

Intermediate output demo:

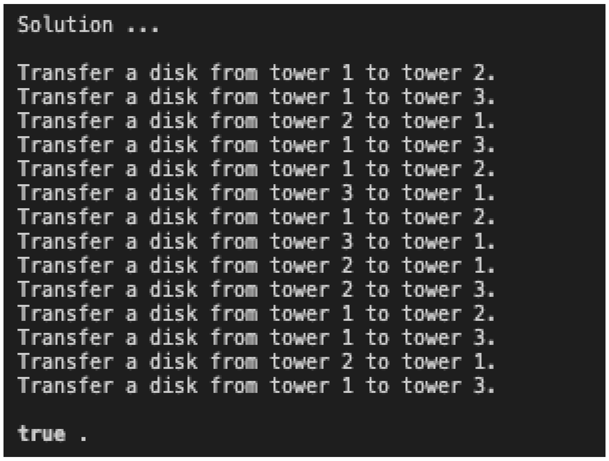


```
?- solve.
PathSoFar’ = ‘[[[s,m,l],[],[]]]
Move’ = ‘m12
NextState’ = ‘[[m,l],[s],[]]
PathSoFar’ = ‘[[[s,m,l],[],[]],[[m,l],[s],[]]]
Move’ = ‘m12
NextState’ = ‘[[l],[m,s],[]]
Move’ = ‘m13
NextState’ = ‘[[l],[s],[m]]
PathSoFar’ = ‘[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]]]
Move’ = ‘m12
NextState’ = ‘[[],[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
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
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' = '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]
```

English output demo:



```
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 .
```

Questions:

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

   The length was 14 moves.

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

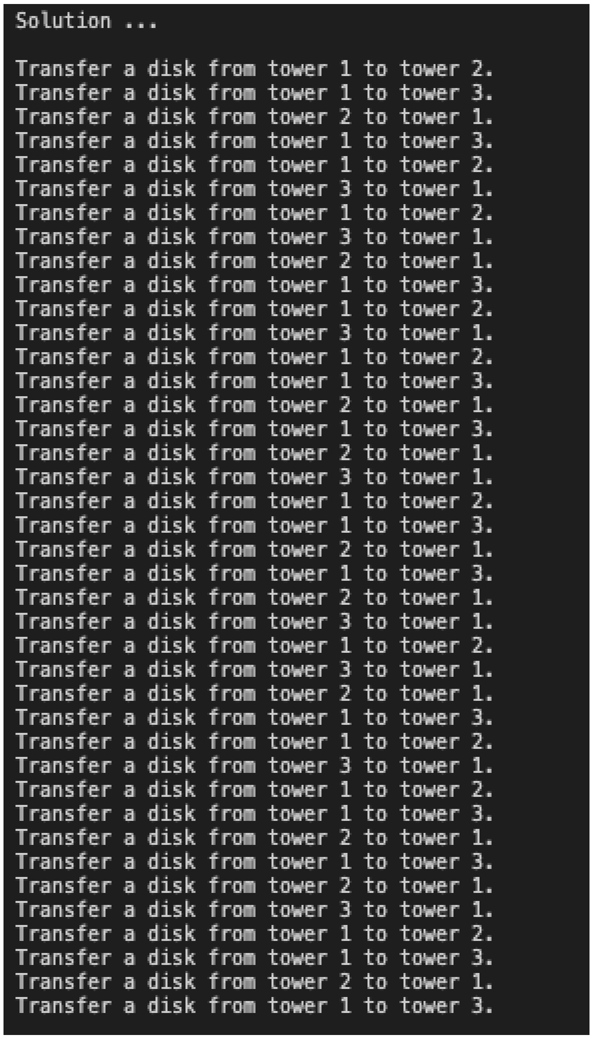
   The length is 7 moves.

3. **How do you account for the discrepancy?**

   The program is testing all different paths to reach a solution whereas humans can do it fairly easily in their heads.

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

English output demo:



```
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 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 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.
```

Questions:

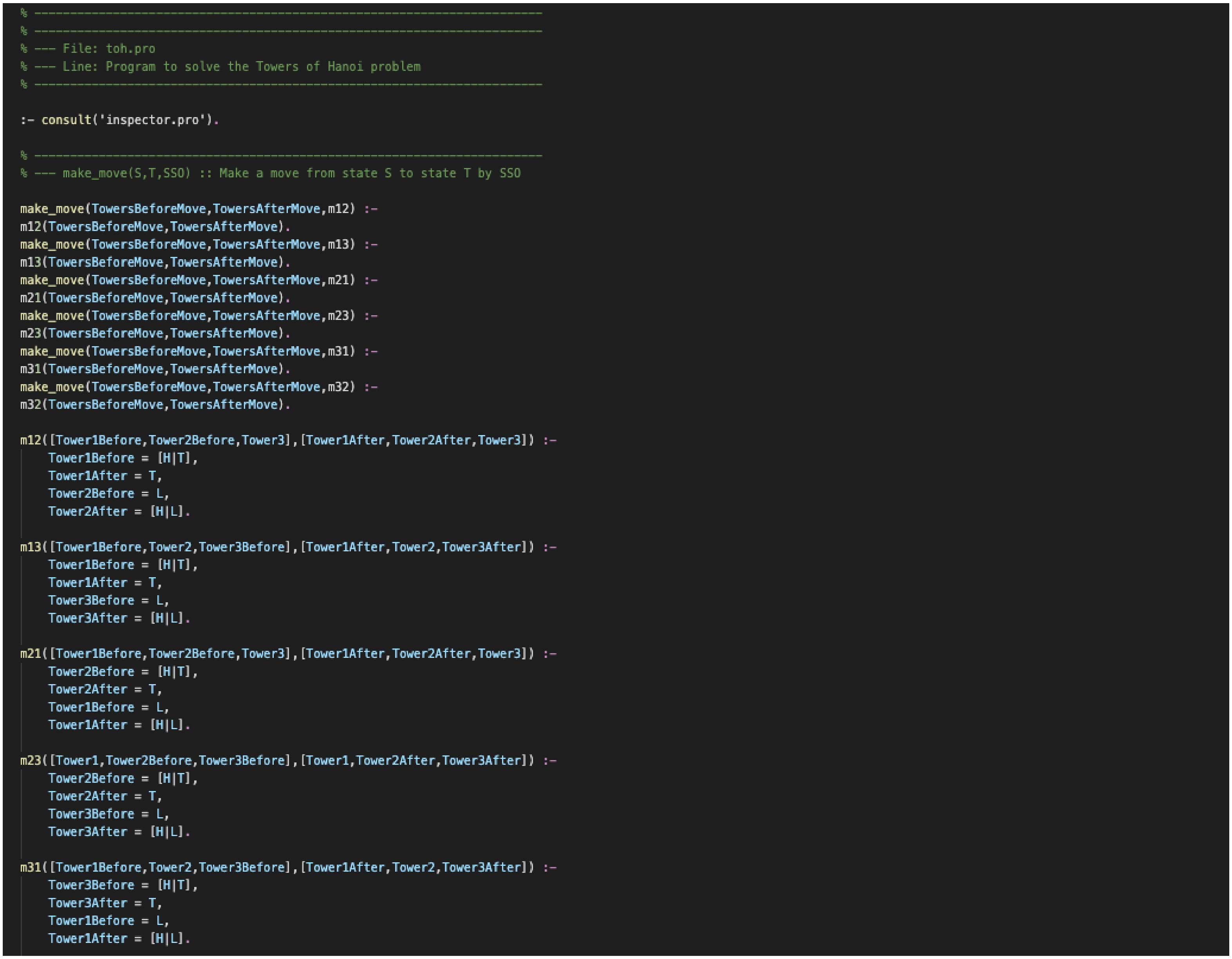
1. **What was the length of your program’s solution to the four disk problem?**

   The length was 40 moves.

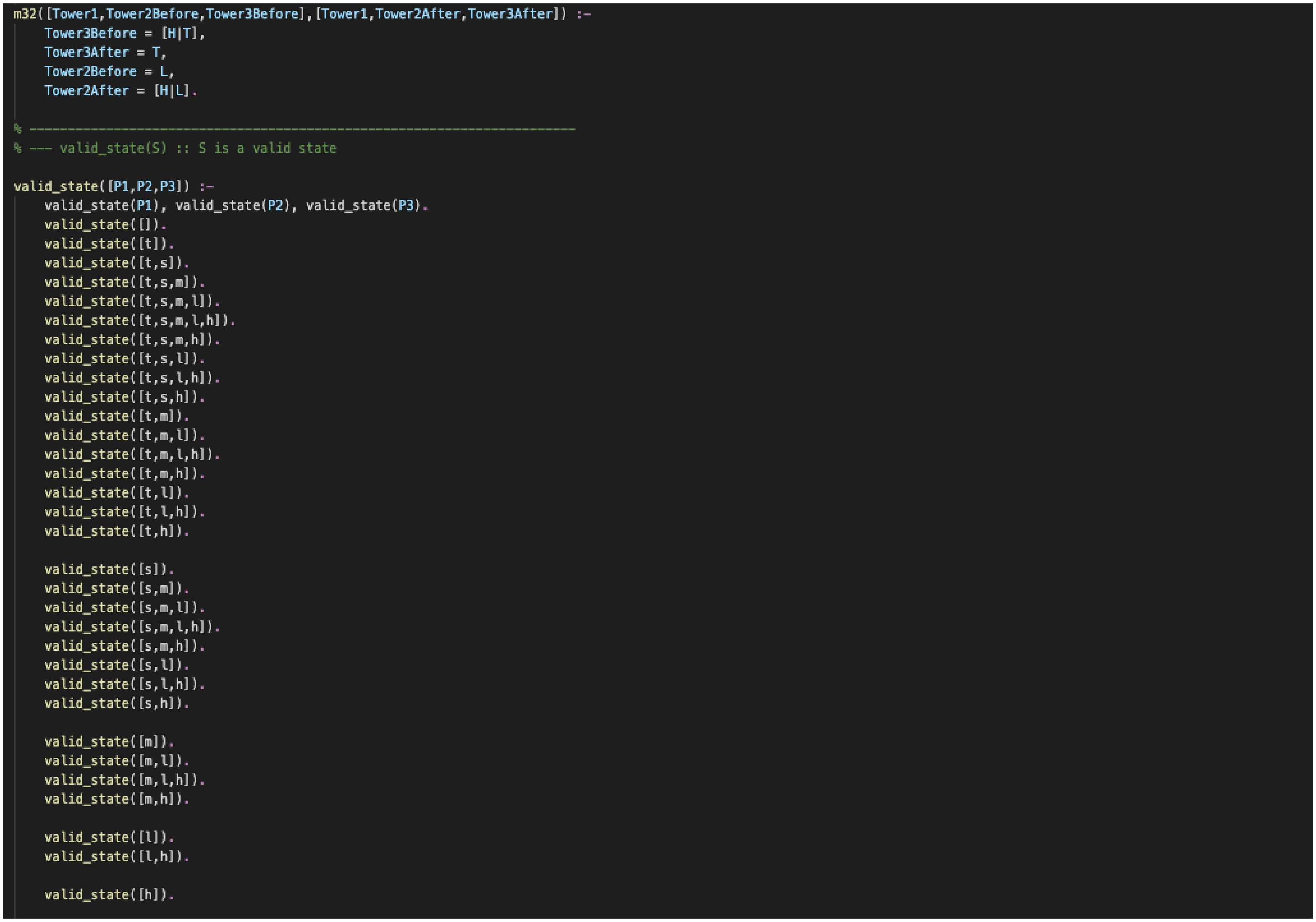
2. **What is the length of the shortest solution to the four disk problem?**

   The length is 15 moves.

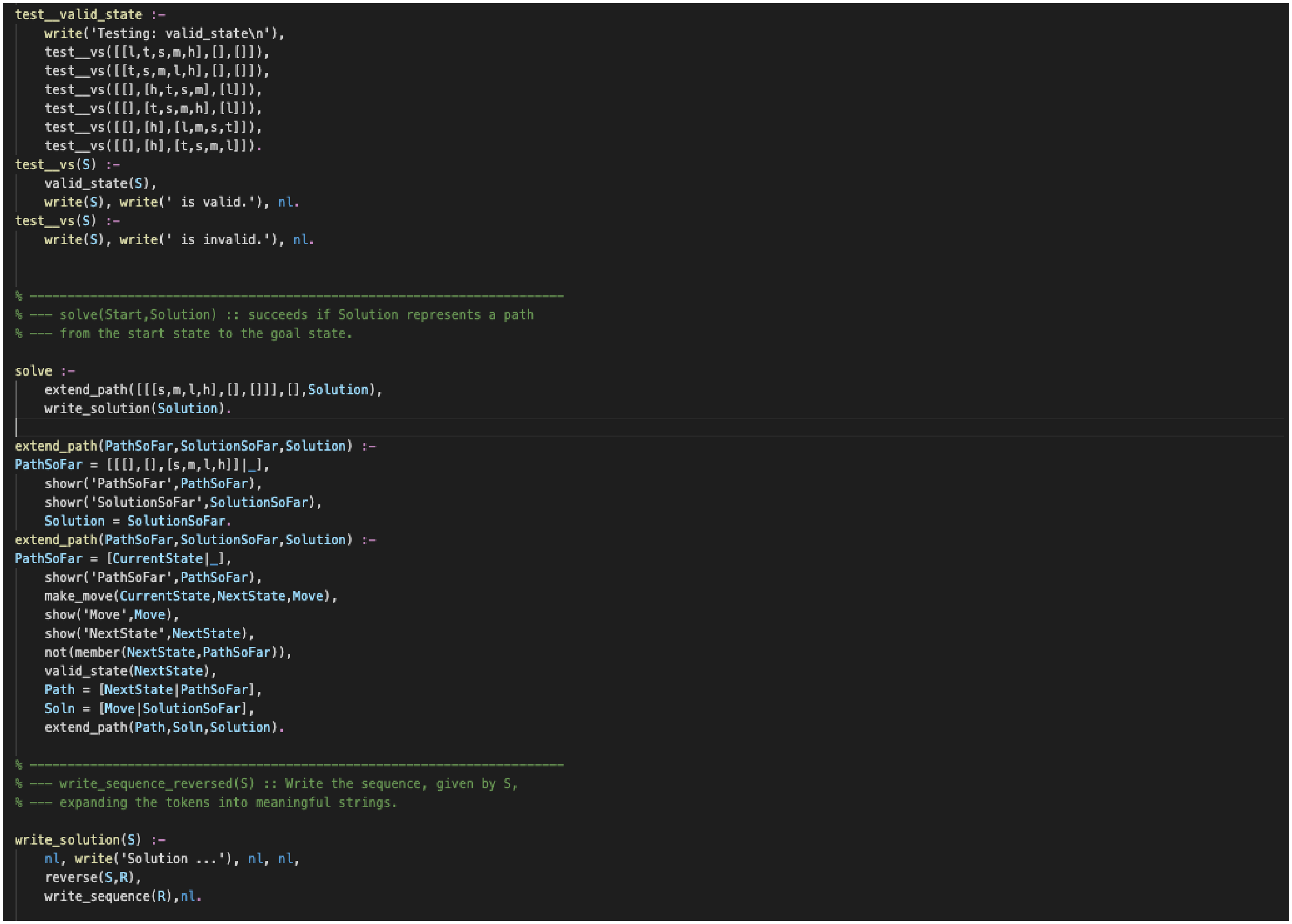
Task 9: Review the Code and Archive It



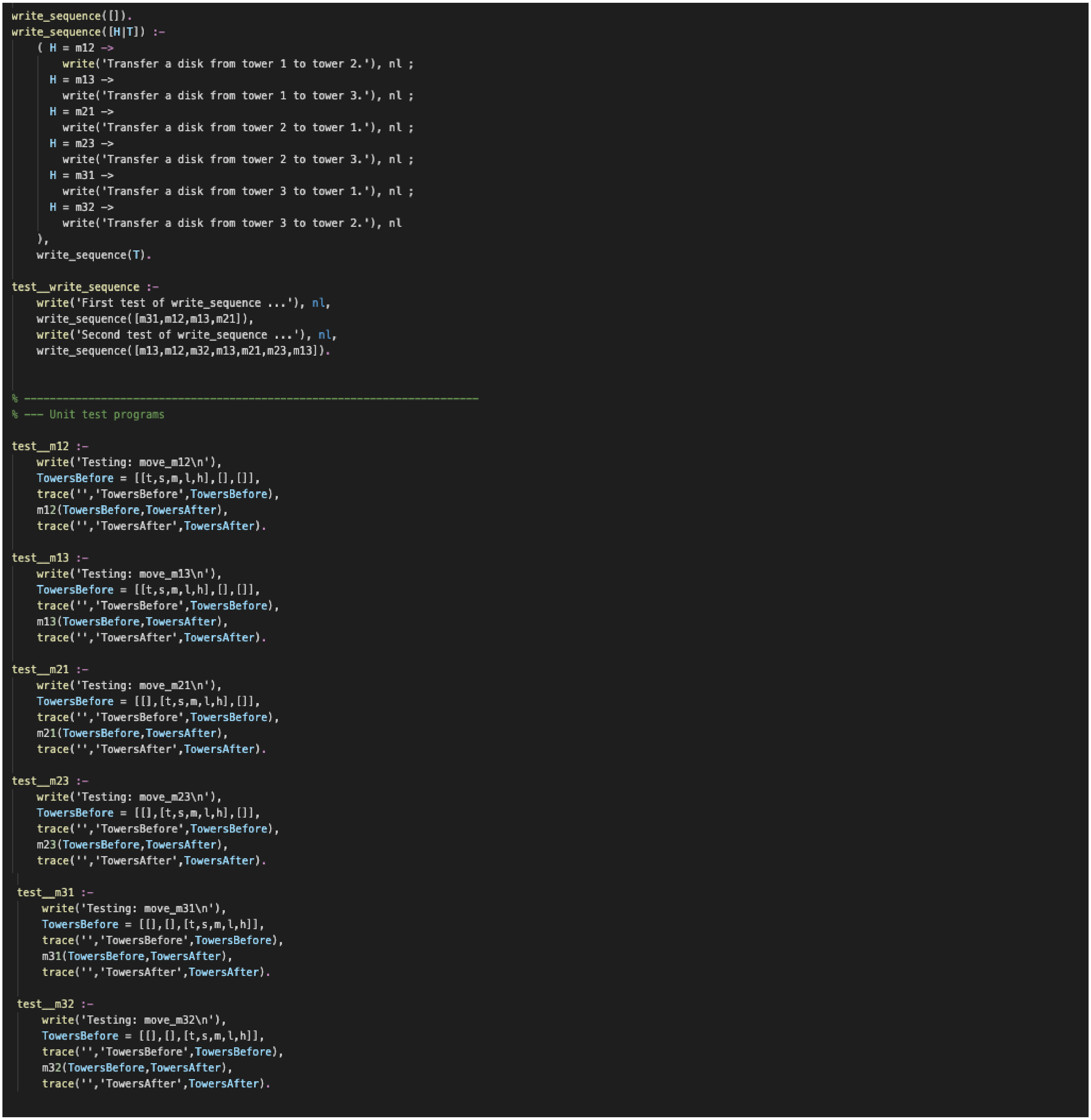
```
% -----------------------------------------------------------------------
% -----------------------------------------------------------------------
% --- File: toh.pro
% --- Line: Program to solve the Towers of Hanoi problem
% -----------------------------------------------------------------------

:- 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,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([P1,P2,P3]) :-
    valid_state(P1), valid_state(P2), valid_state(P3).
    valid_state([]).
    valid_state([t]).
    valid_state([t,s]).
    valid_state([t,s,m]).
    valid_state([t,s,m,l]).
    valid_state([t,s,m,l,h]).
    valid_state([t,s,m,h]).
    valid_state([t,s,l]).
    valid_state([t,s,l,h]).
    valid_state([t,s,h]).
    valid_state([t,m]).
    valid_state([t,m,l]).
    valid_state([t,m,l,h]).
    valid_state([t,m,h]).
    valid_state([t,l]).
    valid_state([t,l,h]).
    valid_state([t,h]).

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

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

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

    valid_state([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.



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

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).
```