Nathaniel Wolf
Prolog Programming Assignment #2: State Space Problem Solving
4/19/2022
CSC 344

Learning Abstract:

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

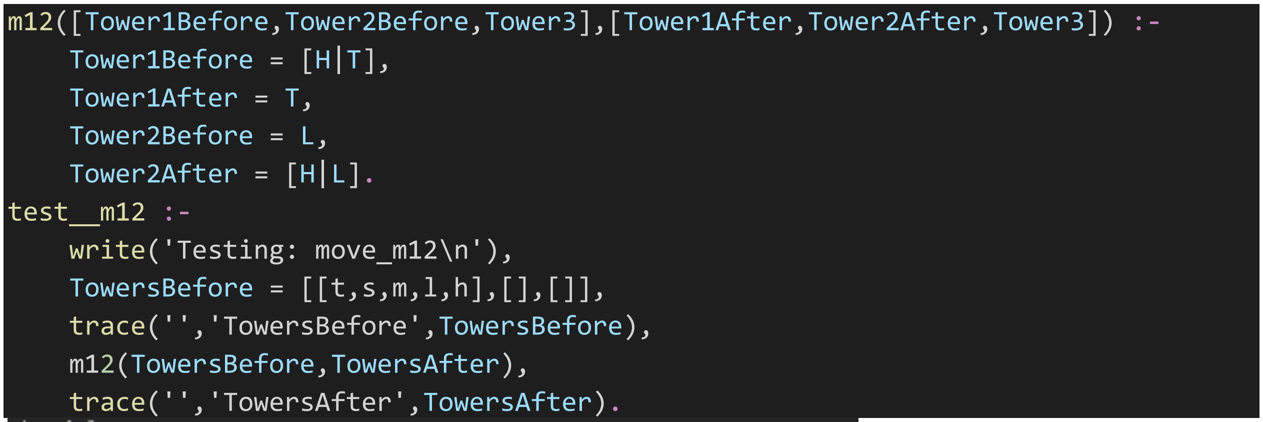
Task 1:

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

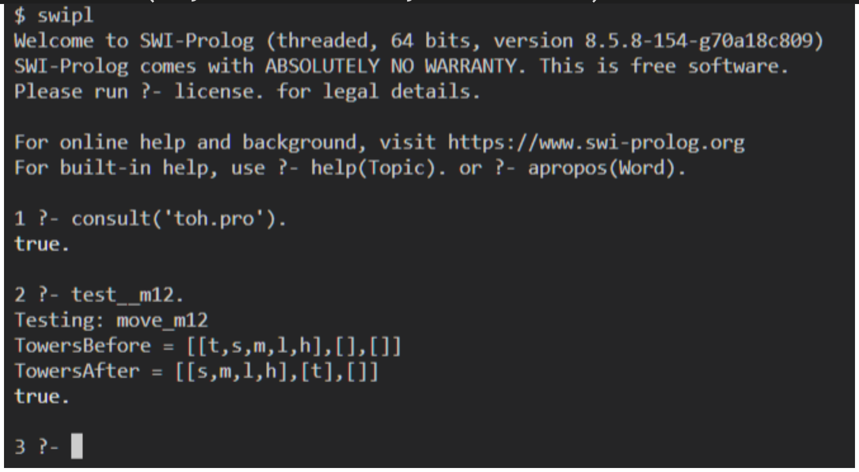
Task 2:

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

Task 3: One Move Predicate and a Unit Test



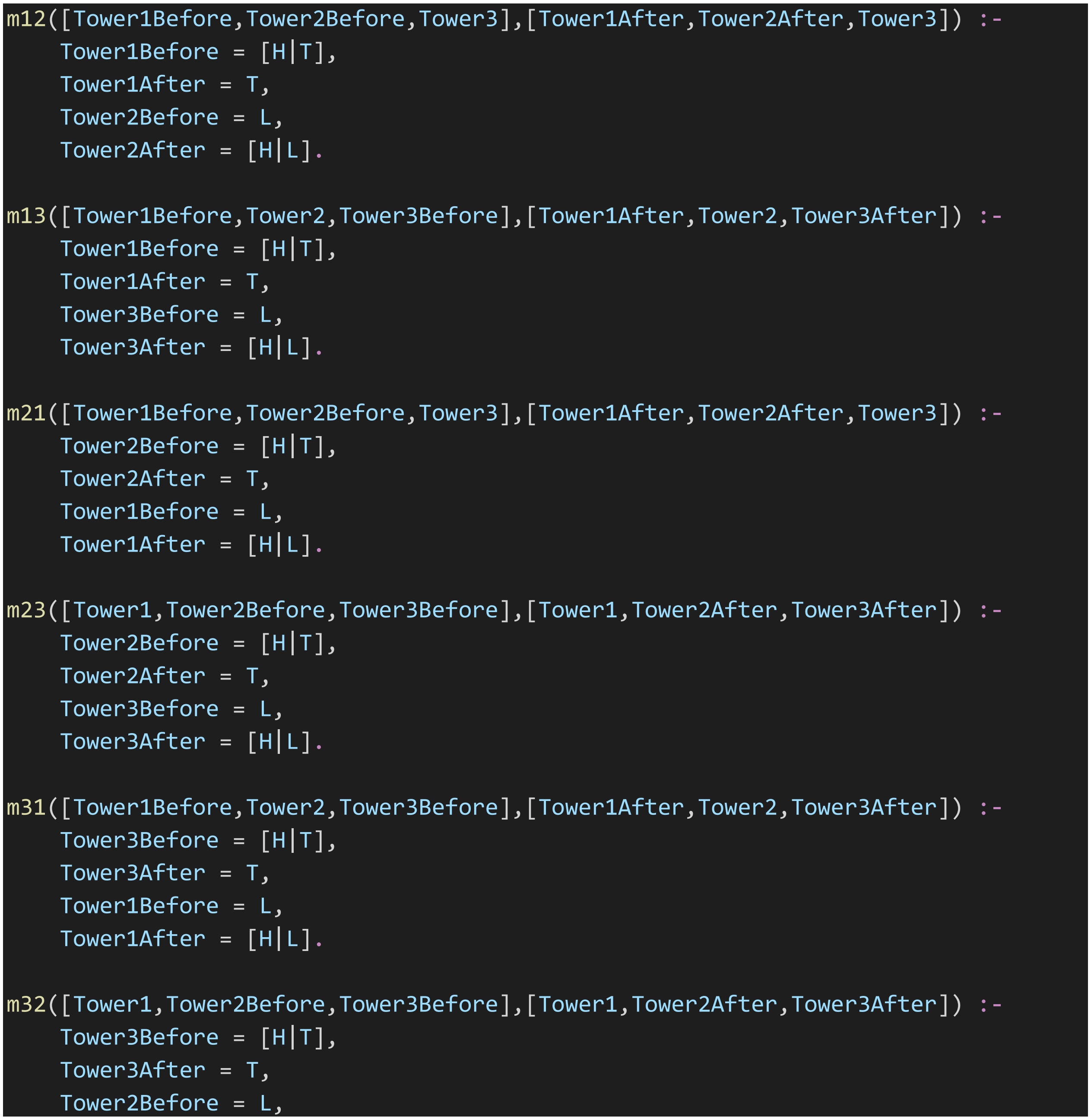
```
m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
    Tower1Before = [H|T],
    Tower1After = T,
    Tower2Before = L,
    Tower2After = [H|L].
test__m12 :-
    write('Testing: move_m12\n'),
    TowersBefore = [[t,s,m,l,h],[],[]],
    trace('','TowersBefore',TowersBefore),
    m12(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).
```



```
$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.5.8-154-g70a18c809)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

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

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

3 ?- 
```

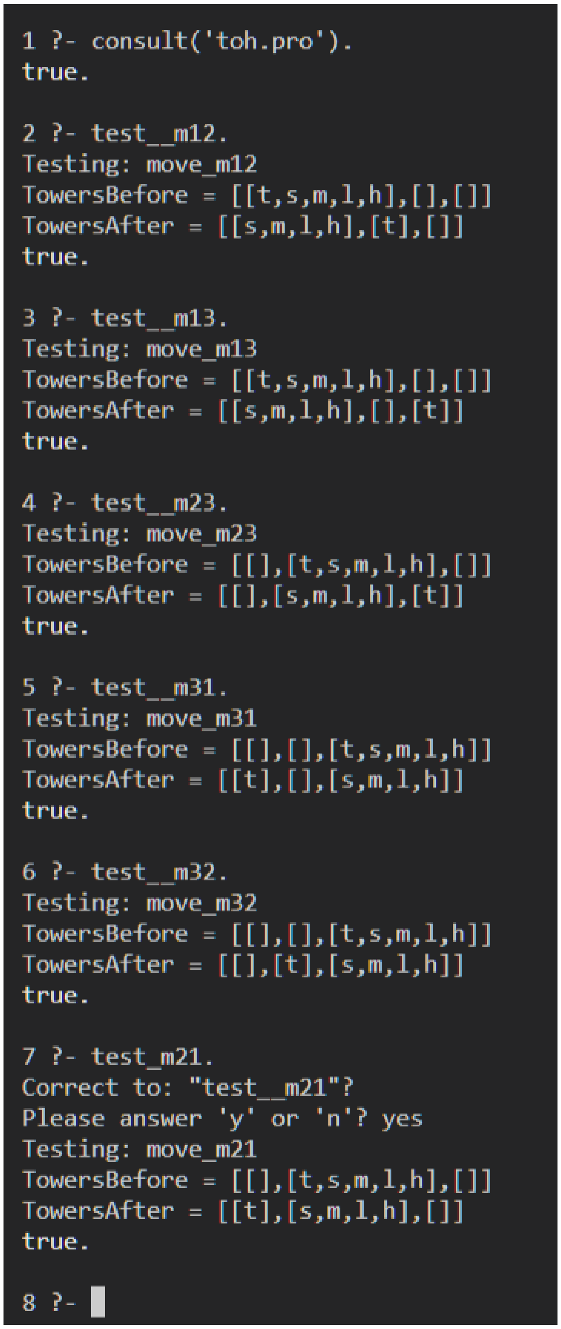
Nathaniel Wolf
Prolog Programming Assignment #2: State Space Problem Solving
4/19/2022
CSC 344

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



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

m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-
    Tower1Before = [H|T],
    Tower1After = T,
    Tower3Before = L,
    Tower3After = [H|L].

m21([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
    Tower2Before = [H|T],
    Tower2After = T,
    Tower1Before = L,
    Tower1After = [H|L].

m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-
    Tower2Before = [H|T],
    Tower2After = T,
    Tower3Before = L,
    Tower3After = [H|L].

m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-
    Tower3Before = [H|T],
    Tower3After = T,
    Tower1Before = L,
    Tower1After = [H|L].

m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-
    Tower3Before = [H|T],
    Tower3After = T,
    Tower2Before = L,
    Tower2After = [H|L].
```

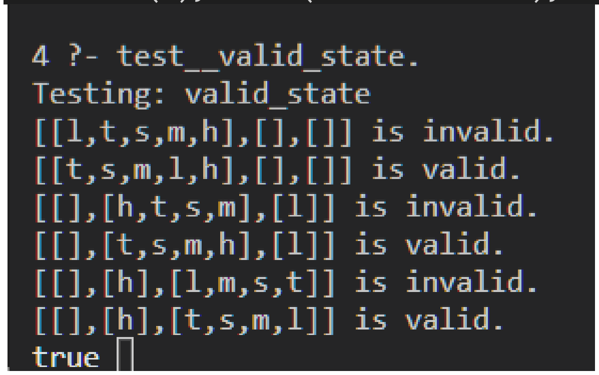
```
% --- Unit test programs

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

test__m13 :-
    write('Testing: move_m13\n'),
    TowersBefore = [[t,s,m,l,h],[],[]],
    trace('','TowersBefore',TowersBefore),
    m13(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__m21 :-
    write('Testing: move_m21\n'),
    TowersBefore = [[],[t,s,m,l,h],[]],
    trace('','TowersBefore',TowersBefore),
    m21(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__m23 :-
    write('Testing: move_m23\n'),
    TowersBefore = [[],[t,s,m,l,h],[]],
    trace('','TowersBefore',TowersBefore),
    m23(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__m31 :-
    write('Testing: move_m31\n'),
    TowersBefore = [[],[],[t,s,m,l,h]],
    trace('','TowersBefore',TowersBefore),
    m31(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__m32 :-
    write('Testing: move_m32\n'),
    TowersBefore = [[],[],[t,s,m,l,h]],
    trace('','TowersBefore',TowersBefore),
    m32(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).
```



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

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

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

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

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

6 ?- test__m32.
Testing: move_m32
TowersBefore = [[],[],[t,s,m,l,h]]
TowersAfter = [[],[t],[s,m,l,h]]
true.

7 ?- test_m21.
Correct to: "test__m21"?
Please answer 'y' or 'n'? yes
Testing: move_m21
TowersBefore = [[],[t,s,m,l,h],[]]
TowersAfter = [[t],[s,m,l,h],[]]
true.

8 ?- 
```

Task 5: Valid State Predicate and Unit Test

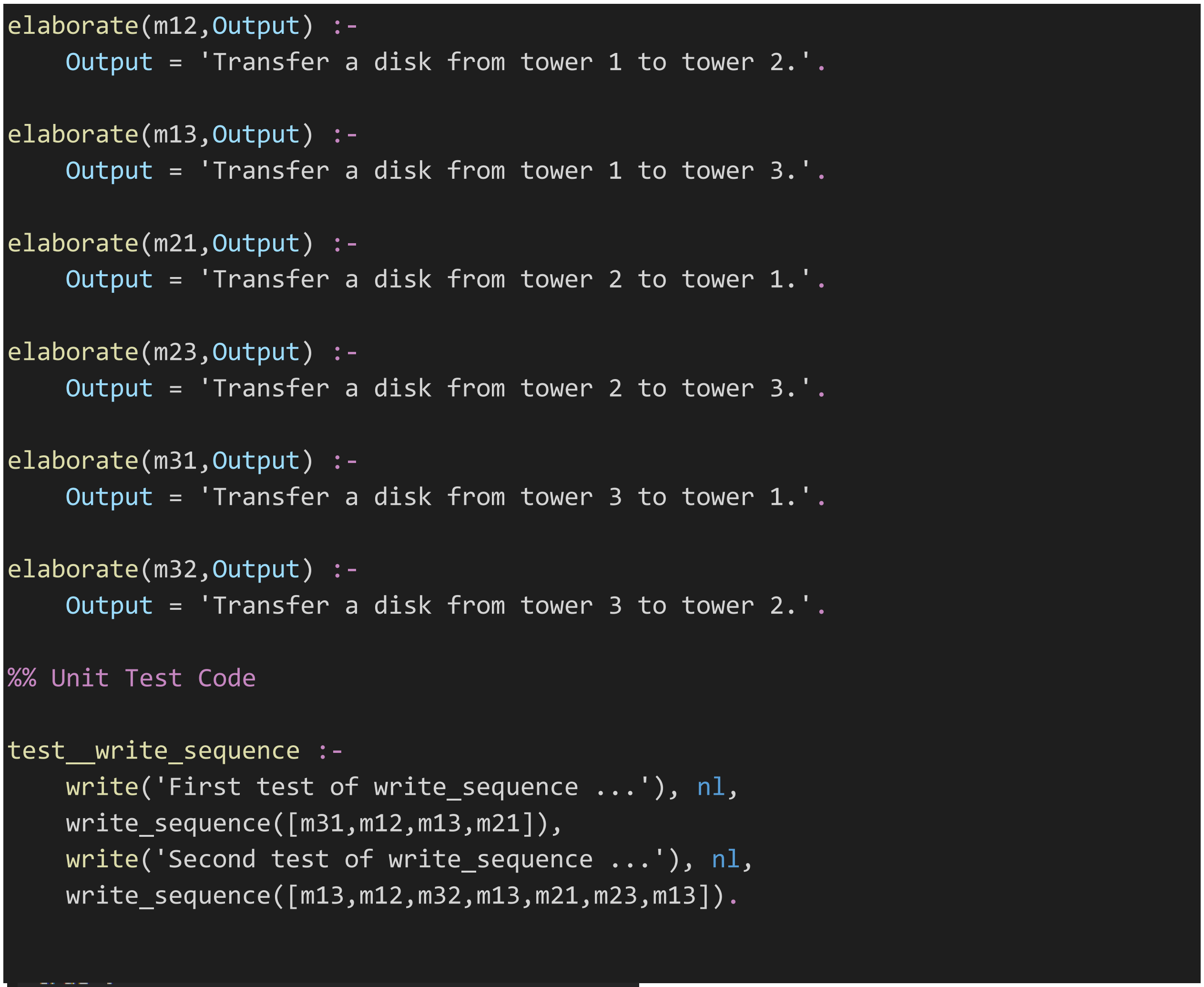
```
% ---------------------------------------------------------------------
% --- valid_state(S) :: S is a valid state

valid_state([A|[B|[C]]]) :- towerState(A), towerState(B), towerState(C).

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

%% Unit Test Code


test__valid_state :-
    write('Testing: valid_state\n'),
    test__vs([[l,t,s,m,h],[],[]]),
    test__vs([[t,s,m,l,h],[],[]]),
    test__vs([[],[h,t,s,m],[l]]),
    test__vs([[],[t,s,m,h],[l]]),
    test__vs([[],[h],[l,m,s,t]]),
    test__vs([[],[h],[t,s,m,l]]).

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

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



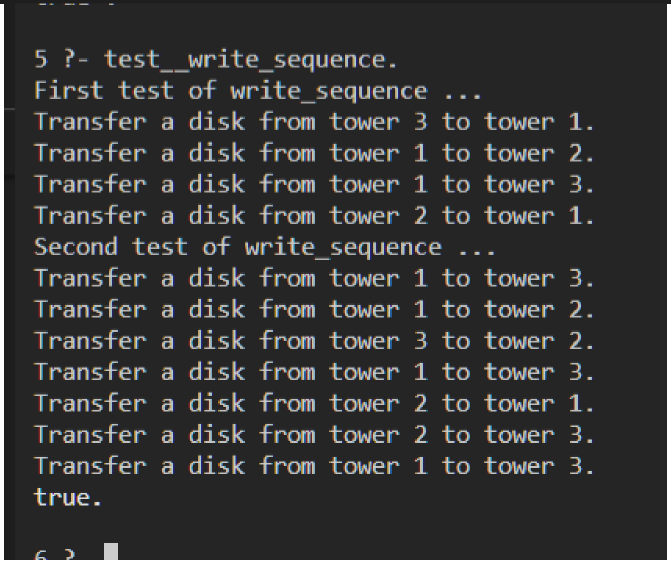
```
4 ?- test__valid_state.
Testing: valid_state
[[l,t,s,m,h],[],[]] is invalid.
[[t,s,m,l,h],[],[]] is valid.
[[],[h,t,s,m],[l]] is invalid.
[[],[t,s,m,h],[l]] is valid.
[[],[h],[l,m,s,t]] is invalid.
[[],[h],[t,s,m,l]] is valid.
true
```

Task 6: Defining the write sequence predicate

```
%% Write Sequence Doe

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

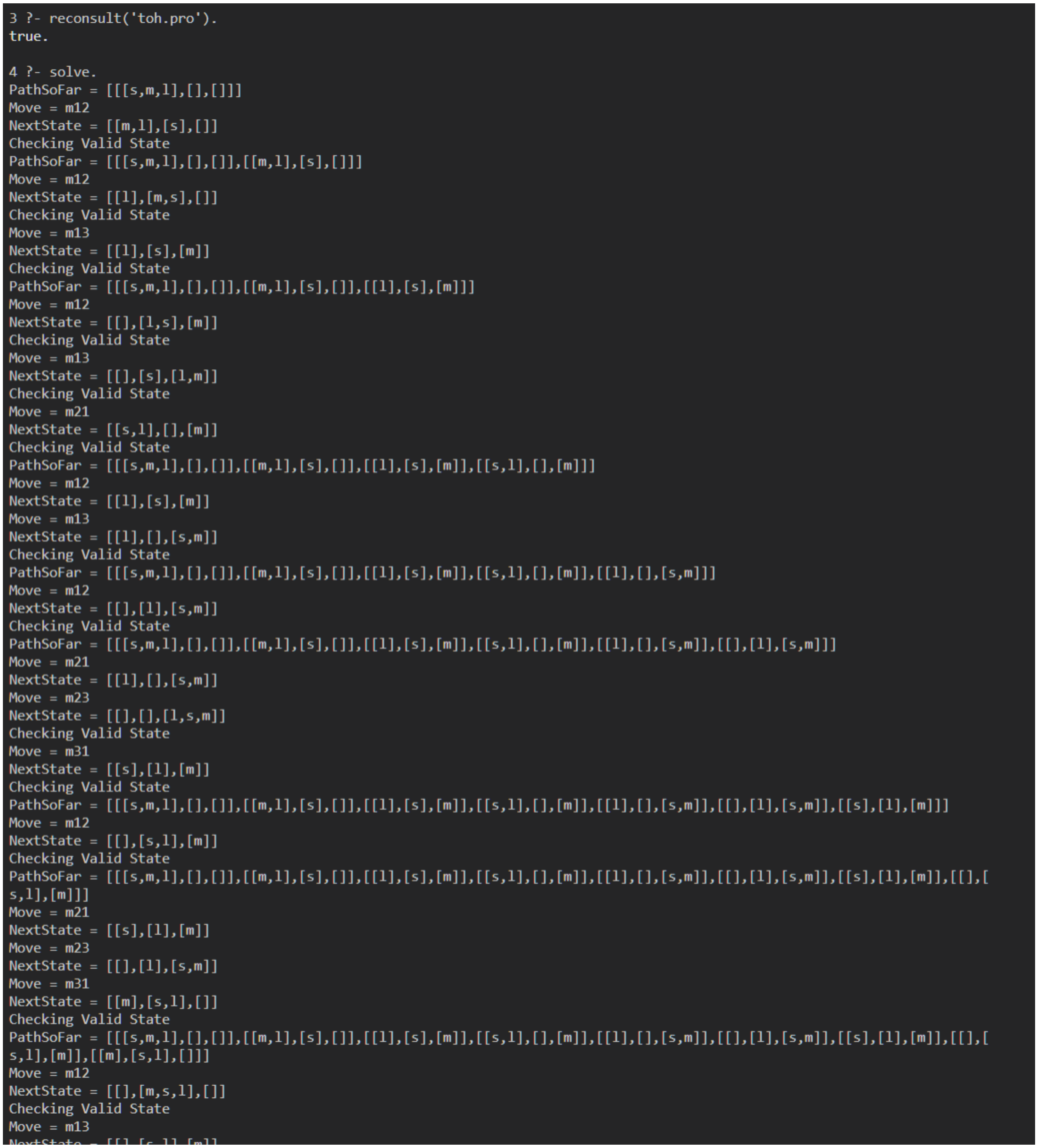


```
elaborate(m12,Output) :-
    Output = 'Transfer a disk from tower 1 to tower 2.'.

elaborate(m13,Output) :-
    Output = 'Transfer a disk from tower 1 to tower 3.'.

elaborate(m21,Output) :-
    Output = 'Transfer a disk from tower 2 to tower 1.'.

elaborate(m23,Output) :-
    Output = 'Transfer a disk from tower 2 to tower 3.'.

elaborate(m31,Output) :-
    Output = 'Transfer a disk from tower 3 to tower 1.'.

elaborate(m32,Output) :-
    Output = 'Transfer a disk from tower 3 to tower 2.'.

%% Unit Test Code

test__write_sequence :-
    write('First test of write_sequence ...'), nl,
    write_sequence([m31,m12,m13,m21]),
    write('Second test of write_sequence ...'), nl,
    write_sequence([m13,m12,m32,m13,m21,m23,m13]).
```

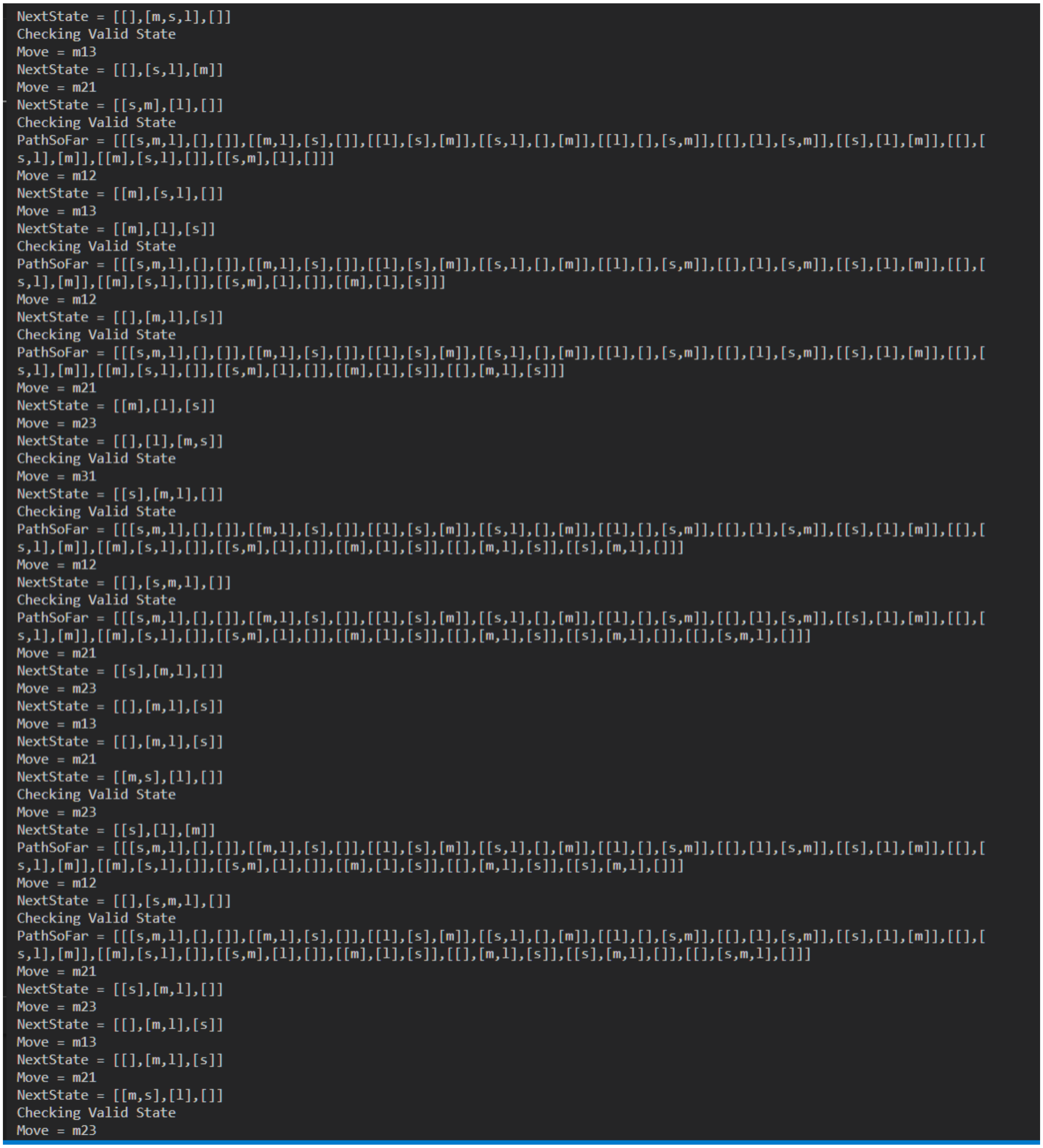


```
5 ?- test__write_sequence.
First test of write_sequence ...
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Second test of write_sequence ...
Transfer a disk from tower 1 to tower 3.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

6 ?-
```

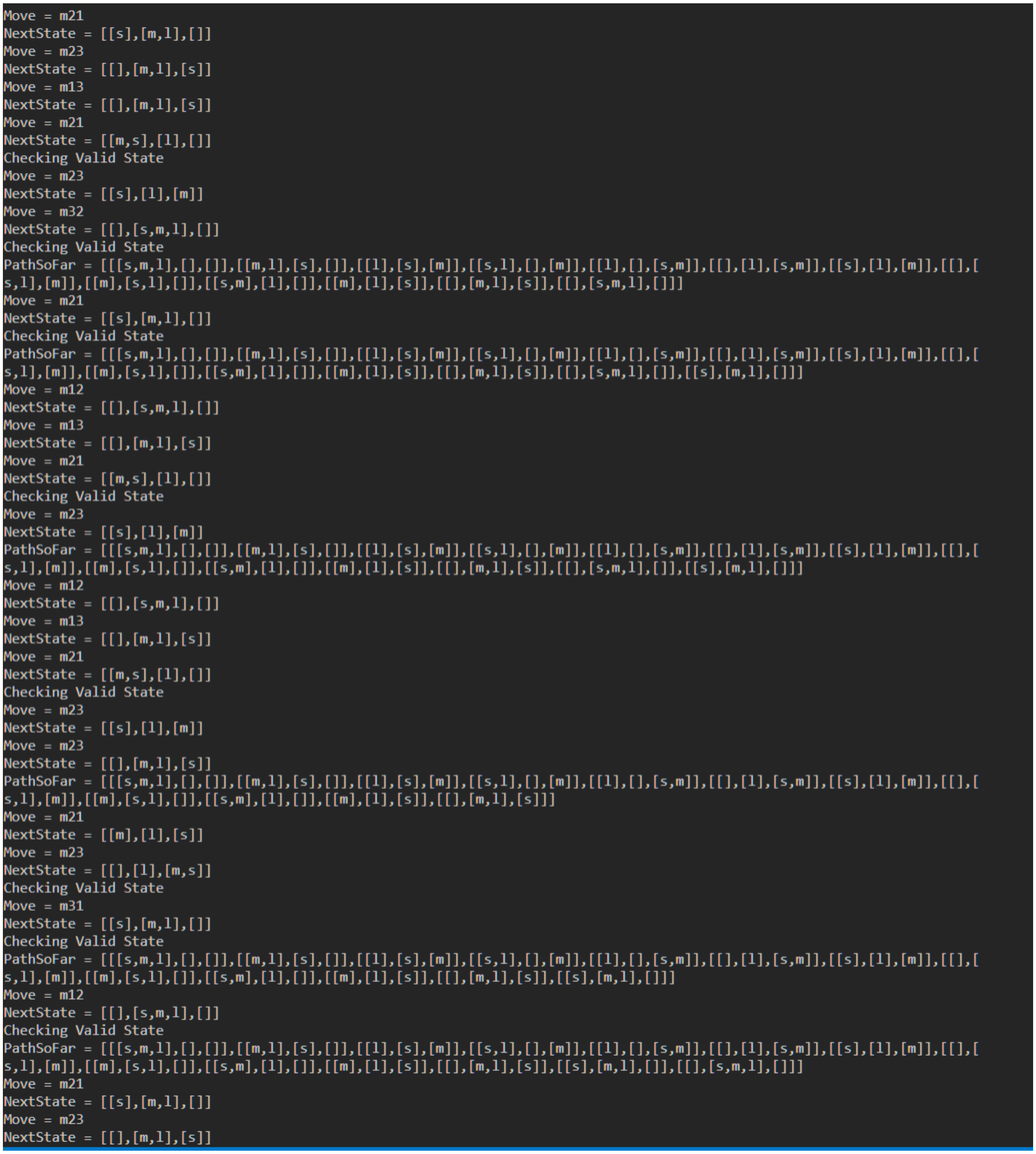
Task 7: Intermediate and Plain English Demo



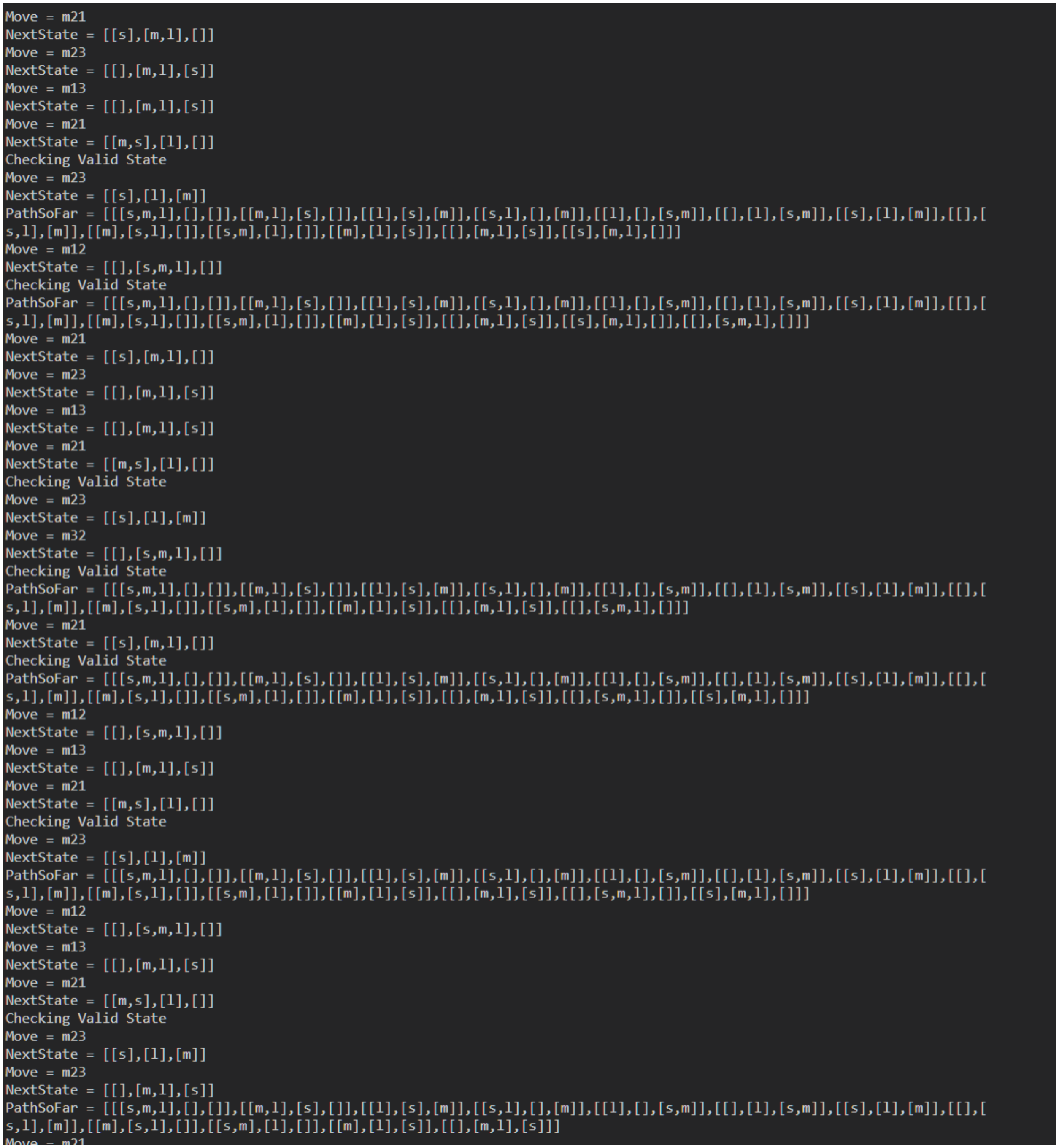
```
3 ?- reconsult('toh.pro').
true.

4 ?- solve.
PathSoFar = [[[s,m,l],[],[]]]
Move = m12
NextState = [[m,l],[s],[]]
Checking Valid State
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]]]
Move = m12
NextState = [[l],[m,s],[]]
Checking Valid State
Move = m13
NextState = [[l],[s],[m]]
Checking Valid State
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]]]
Move = m12
NextState = [[],[l,s],[m]]
Checking Valid State
Move = m13
NextState = [[],[s],[l,m]]
Checking Valid State
Move = m21
NextState = [[s,l],[],[m]]
Checking Valid State
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]]
Checking Valid State
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]]]
Move = m12
NextState = [[],[l],[s,m]]
Checking Valid State
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]]
Checking Valid State
Move = m31
NextState = [[s],[l],[m]]
Checking Valid State
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]]
Checking Valid State
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],[]]
Checking Valid State
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],[]]
Checking Valid State
Move = m13
NextState = [[],[s,l],[m]]
```



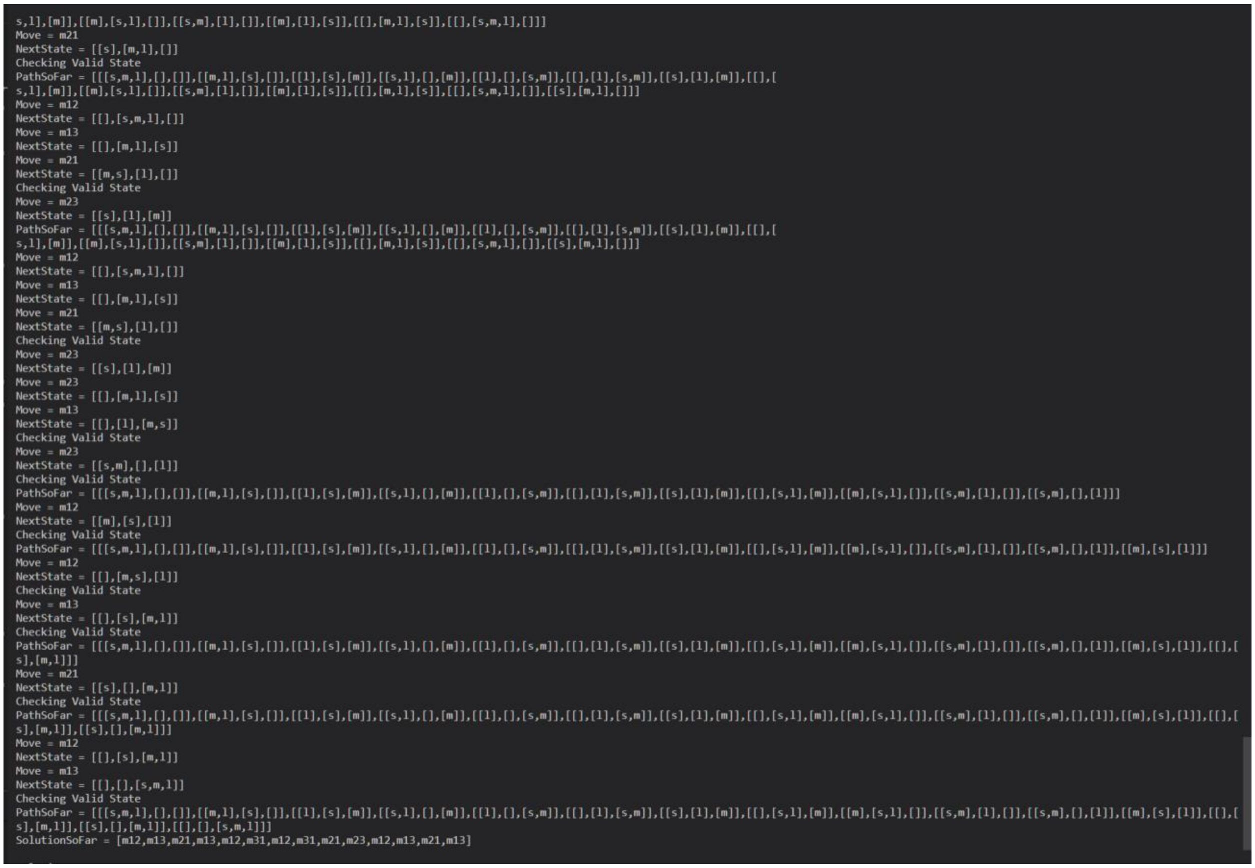
```
NextState = [[],[m,s,l],[]]
Checking Valid State
Move = m13
NextState = [[],[s,l],[m]]
Move = m21
NextState = [[s,m],[l],[]]
Checking Valid State
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[
s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]]]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



```
Move = m21
NextState = [[s],[m,l],[]]
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[m,l],[s]]
Move = m21
NextState = [[m,s],[l],[]]
Checking Valid State
Move = m23
NextState = [[s],[l],[m]]
Move = m32
NextState = [[],[s,m,l],[]]
Checking Valid State
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],[]]
Checking Valid State
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],[]]
Checking Valid State
Move = m23
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]],[[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],[]]
Checking Valid State
Move = m23
NextState = [[s],[l],[m]]
Move = m23
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]]
Checking Valid State
Move = m31
NextState = [[s],[m,l],[]]
Checking Valid State
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],[]]
Checking Valid State
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]]
```

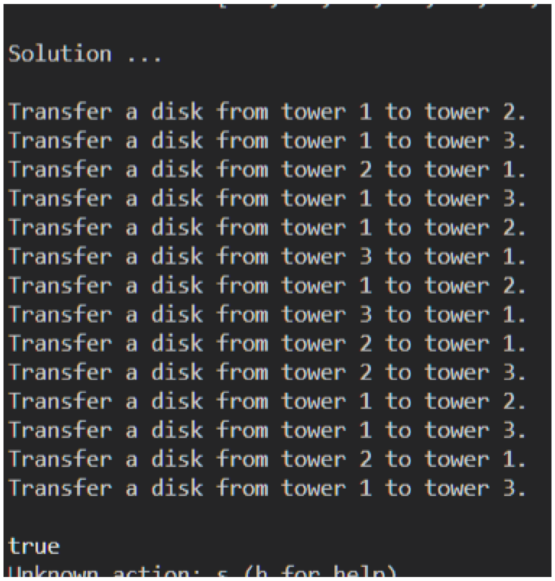


```
Move = m21
NextState = [[s],[m,l],[]]
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[m,l],[s]]
Move = m21
NextState = [[m,s],[l],[]]
Checking Valid State
Move = m23
NextState = [[s],[l],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[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



```
s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[m],[l],[s]],[[],[m,l],[s]],[[],[s,m,l],[]]]
Move = m21
NextState = [[s],[m,l],[]]
Checking Valid State
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],[]]
Checking Valid State
Move = m23
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]],[[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],[]]
Checking Valid State
Move = m23
NextState = [[s],[l],[m]]
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[l],[m,s]]
Checking Valid State
Move = m23
NextState = [[s,m],[],[l]]
Checking Valid State
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]]
Checking Valid State
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]]
Checking Valid State
Move = m13
NextState = [[],[s],[m,l]]
Checking Valid State
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]]
Checking Valid State
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]]
Checking Valid State
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]
```

Paraphrased English Solution



```
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
Unknown action: s (h for help)
```

Nathaniel Wolf
Prolog Programming Assignment #2: State Space Problem Solving
4/19/2022
CSC 344

Questions and Answers

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

The length appears to be a degree of 14 steps.

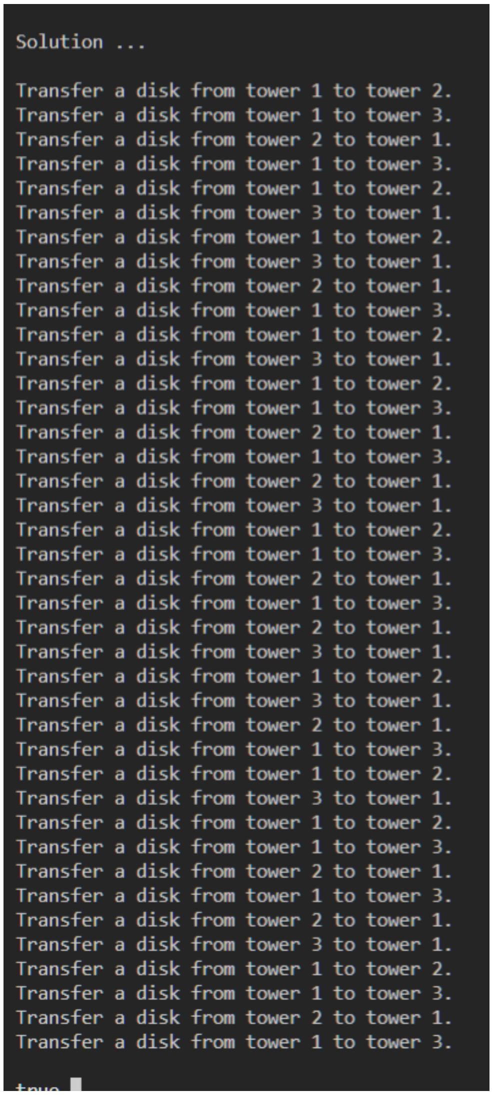
2. *What is the length of the shortest solution to the three-disk problem?*

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

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

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

Task 8:



```
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.

true
```

Nathaniel Wolf
Prolog Programming Assignment #2: State Space Problem Solving
4/19/2022
CSC 344

Questions and Answers

1. *What was the length of your program's solutions to the four-disk problem?*

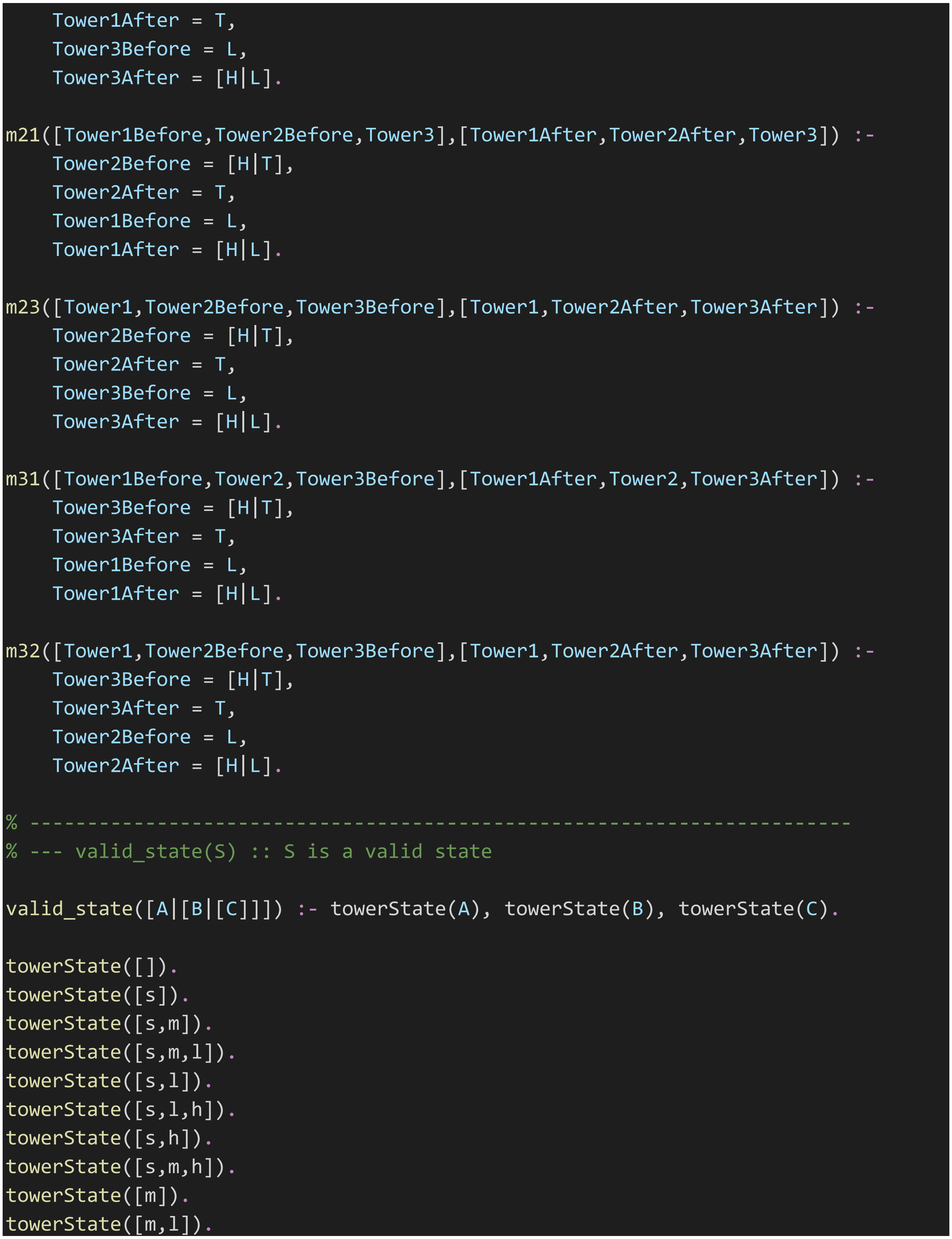
The length appears to be a degree of 40 steps.

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

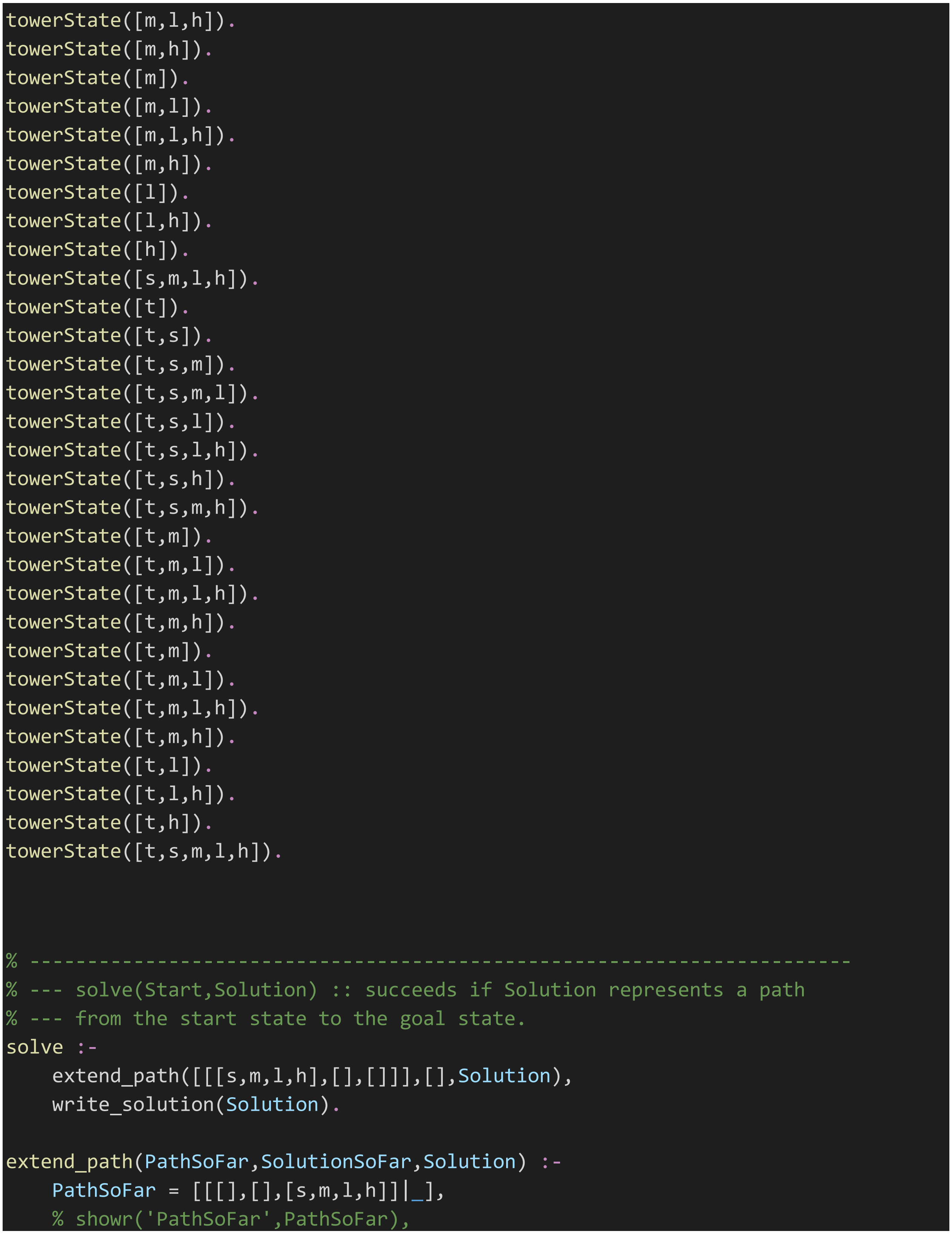
Doing it by hand I was able to complete it in 15 steps with four discs.

Task 9: The Full Code Base

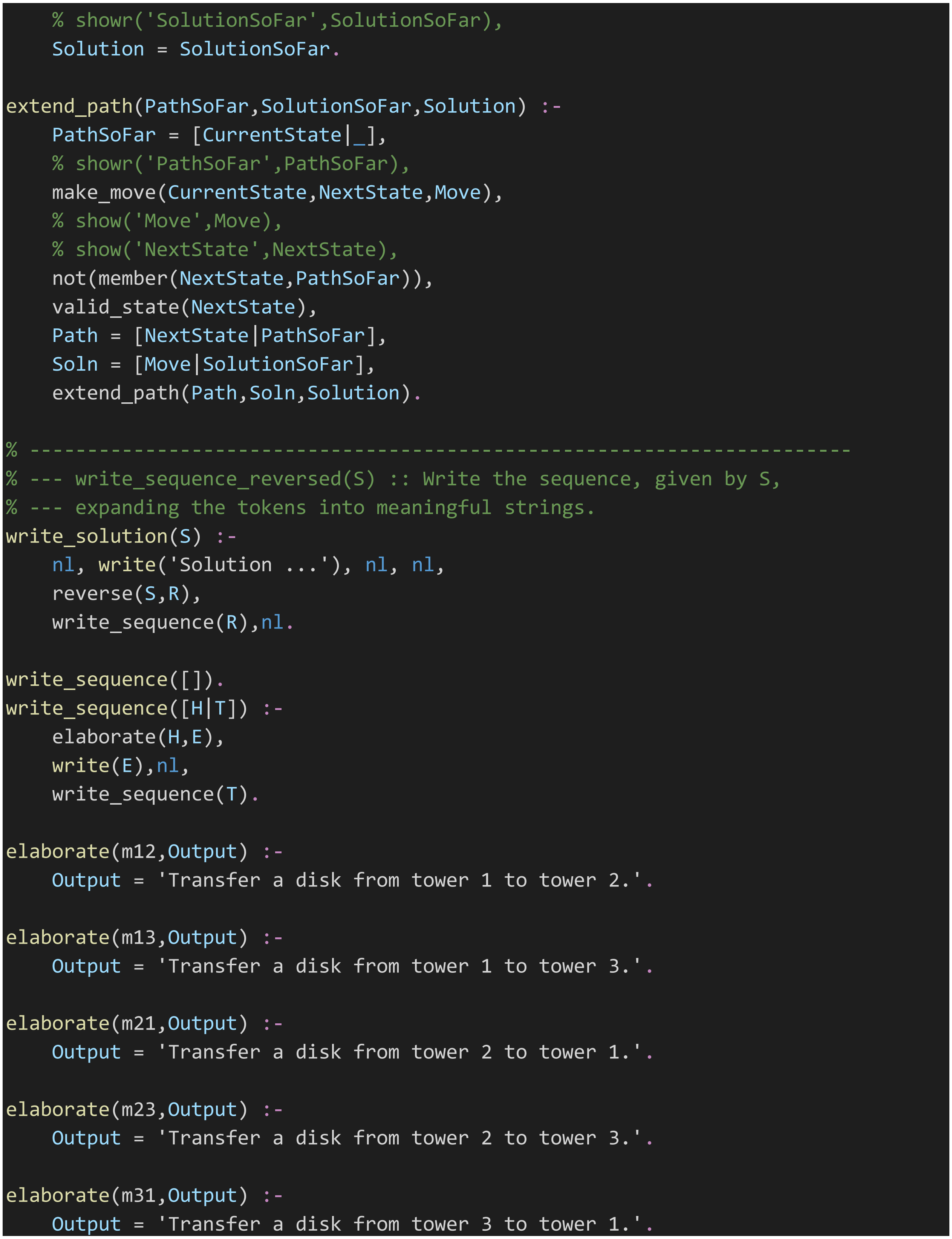
```
% ---------------------------------------------------------------------
% ---------------------------------------------------------------------
% --- File: towers_of_hanoi.pro
% --- Line: Program to solve the Towers of Hanoi problem
% ---------------------------------------------------------------------

:- consult('inspectors.pro').

% ---------------------------------------------------------------------
% --- make_move(S,T,SSO) :: Make a move from state S to state T by SSO
make_move(TowersBeforeMove,TowersAfterMove,m12) :-
    m12(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m13) :-
    m13(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m21) :-
    m21(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m23) :-
    m23(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m31) :-
    m31(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m32) :-
    m32(TowersBeforeMove,TowersAfterMove).

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

m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-
    Tower1Before = [H|T],
```



```prolog
    Tower1After = T,
    Tower3Before = L,
    Tower3After = [H|L].

m21([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
    Tower2Before = [H|T],
    Tower2After = T,
    Tower1Before = L,
    Tower1After = [H|L].

m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-
    Tower2Before = [H|T],
    Tower2After = T,
    Tower3Before = L,
    Tower3After = [H|L].

m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-
    Tower3Before = [H|T],
    Tower3After = T,
    Tower1Before = L,
    Tower1After = [H|L].

m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-
    Tower3Before = [H|T],
    Tower3After = T,
    Tower2Before = L,
    Tower2After = [H|L].

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

valid_state([A|[B|[C]]]) :- towerState(A), towerState(B), towerState(C).

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



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



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

extend_path(PathSoFar,SolutionSoFar,Solution) :-
    PathSoFar = [[[],[],[s,m,l,h]]|_],
    % showr('PathSoFar',PathSoFar),
```



```
    % showr('SolutionSoFar',SolutionSoFar),
    Solution = SolutionSoFar.

extend_path(PathSoFar,SolutionSoFar,Solution) :-
    PathSoFar = [CurrentState|_],
    % showr('PathSoFar',PathSoFar),
    make_move(CurrentState,NextState,Move),
    % show('Move',Move),
    % show('NextState',NextState),
    not(member(NextState,PathSoFar)),
    valid_state(NextState),
    Path = [NextState|PathSoFar],
    Soln = [Move|SolutionSoFar],
    extend_path(Path,Soln,Solution).

% ----------------------------------------------------------------------
% --- write_sequence_reversed(S) :: Write the sequence, given by S,
% --- expanding the tokens into meaningful strings.
write_solution(S) :-
    nl, write('Solution ...'), nl, nl,
    reverse(S,R),
    write_sequence(R),nl.

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

elaborate(m12,Output) :-
    Output = 'Transfer a disk from tower 1 to tower 2.'.

elaborate(m13,Output) :-
    Output = 'Transfer a disk from tower 1 to tower 3.'.

elaborate(m21,Output) :-
    Output = 'Transfer a disk from tower 2 to tower 1.'.

elaborate(m23,Output) :-
    Output = 'Transfer a disk from tower 2 to tower 3.'.

elaborate(m31,Output) :-
    Output = 'Transfer a disk from tower 3 to tower 1.'.
```

```
elaborate(m32,Output) :-
    Output = 'Transfer a disk from tower 3 to tower 2.'.


% ------------------------------------------------------------------------
% --- Unit test programs

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

test__m12x :-
    write('Testing: move_m12\n'),
    TowersBefore = [[s,m,l,h],[],[t]],
    trace('','TowersBefore',TowersBefore),
    m12(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).


test__m13 :-
    write('Testing: move_m13\n'),
    TowersBefore = [[t,s,m,l,h],[],[]],
    trace('','TowersBefore',TowersBefore),
    m13(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__m13x :-
    write('Testing: move_m13\n'),
    TowersBefore = [[s,m,l,h],[],[t]],
    trace('','TowersBefore',TowersBefore),
    m13(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).


test__m21 :-
    write('Testing: move_m21\n'),
    TowersBefore = [[],[t,s,m,l,h],[]],
    trace('','TowersBefore',TowersBefore),
    m21(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__m23 :-
    write('Testing: move_m23\n'),
    TowersBefore = [[],[t,s,m,l,h],[]],
    trace('','TowersBefore',TowersBefore),
    m23(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__m31 :-
    write('Testing: move_m31\n'),
    TowersBefore = [[],[],[t,s,m,l,h]],
    trace('','TowersBefore',TowersBefore),
    m31(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__m32 :-
    write('Testing: move_m32\n'),
    TowersBefore = [[],[],[t,s,m,l,h]],
    trace('','TowersBefore',TowersBefore),
    m32(TowersBefore,TowersAfter),
    trace('','TowersAfter',TowersAfter).

test__valid_state :-
    write('Testing: valid_state\n'),
    test__vs([[l,t,s,m,h],[],[]]),
    test__vs([[t,s,m,l,h],[],[]]),
    test__vs([[],[h,t,s,m],[l]]),
    test__vs([[],[t,s,m,h],[l]]),
    test__vs([[],[h],[l,m,s,t]]),
    test__vs([[],[h],[t,s,m,l]]).

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

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

test__write_sequence :-
    write('First test of write_sequence ...'), nl,
    write_sequence([m31,m12,m13,m21]),
    write('Second test of write_sequence ...'), nl,
    write_sequence([m13,m12,m32,m13,m21,m23,m13]).
```