# Unit 4 Seminar: Introduction to Logic Programming

1. Prolog can be used to test the questions included in Unit 2. For example, to test exercise 1 carry out the following steps.

    - Surf to https://swi-prolog.org
    - Click on “try swi-prolog online”.
    - On the SWISH page click on notebook.
    - Click on Query.
    - In the ‘query’ box enter “member(c, [a,b,c,2,3,4])”.
    - Click the go (>) button – it should give the answer ‘true’ (I.e., c is a member of the set).
    - How many of the questions in exercise 1 can you check in this way?

Given the following sets:

A = {a,b,c,2,3,4}

B = {a, b}

C = {c,2}

D = {b,c}

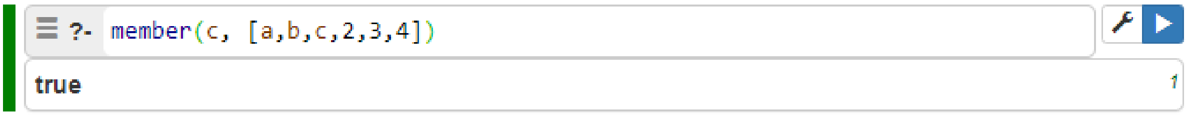
E = {a,b,{c}}

F = ∅

G = {{a, b},{c, 2}}

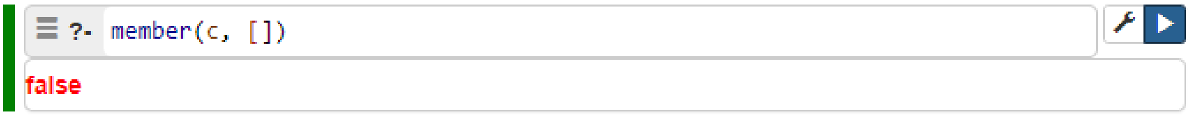
classify each of the following statements as true or false

(a) c ∈ A TRUE



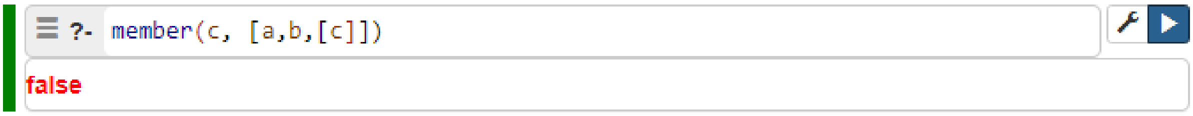
```
?- member(c, [a,b,c,2,3,4])
true
```

(b) c ∈ F FALSE



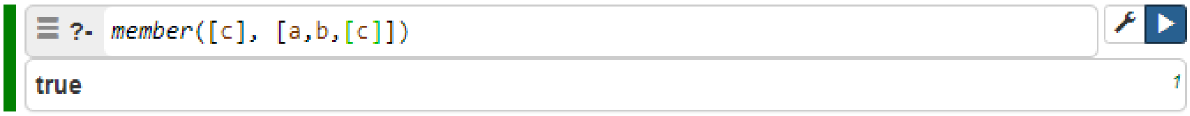
```
?- member(c, [])
false
```

(c) c ∈ E FALSE



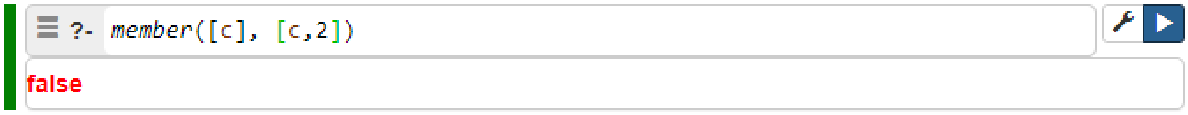
```
?- member(c, [a,b,[c]])
false
```

(d) {c} ∈ E TRUE



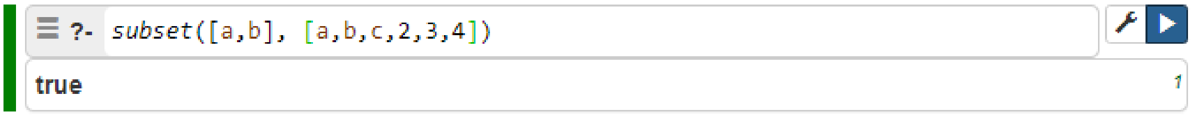
```
?- member([c], [a,b,[c]])
true
```

(e) {c} ∈ C FALSE



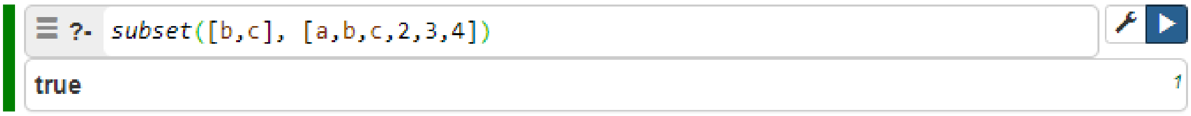
```
?- member([c], [c,2])
false
```

(f) B ⊆ A TRUE



```
?- subset([a,b], [a,b,c,2,3,4])
true
```

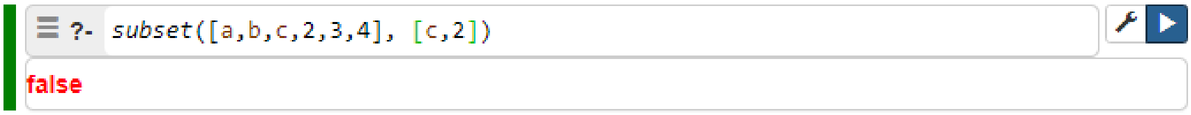
(g) D ⊂ A TRUE



```
?- subset([b,c], [a,b,c,2,3,4])
true
```

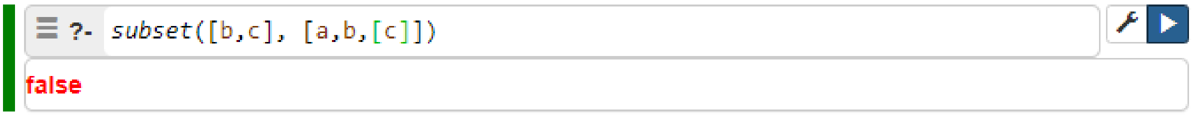
However this used the subset command. There is no **proper** subset feature in Prolog so it's a lucky coincidence that A does not also contain D as a subset

(h) A ⊆ C FALSE



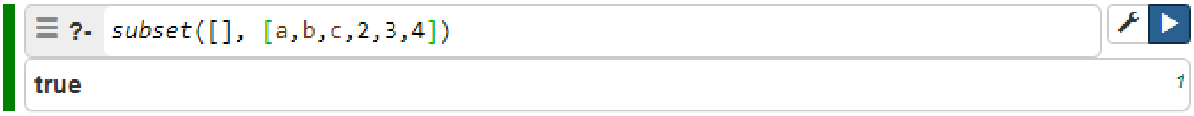
```
?- subset([a,b,c,2,3,4], [c,2])
false
```

(i) D ⊆ E FALSE



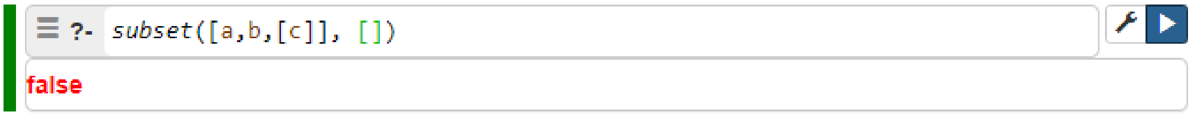
```
?- subset([b,c], [a,b,[c]])
false
```

(j) F ⊆ A TRUE



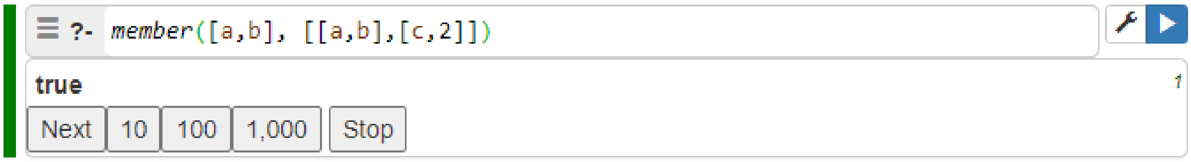
```
?- subset([], [a,b,c,2,3,4])
true
```

(k) E ⊆ F FALSE



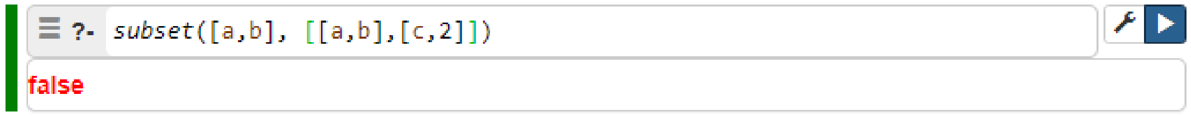
```
?- subset([a,b,[c]], [])
false
```

(l) B ∈ G TRUE



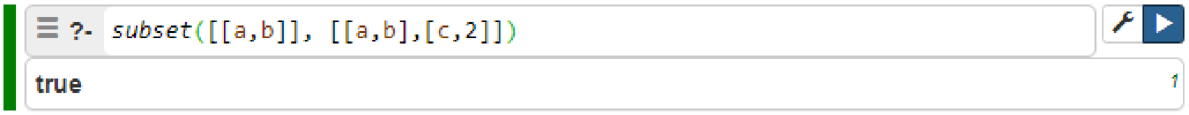
```
?- member([a,b], [[a,b],[c,2]])
true
```

(m) B ⊆ G FALSE



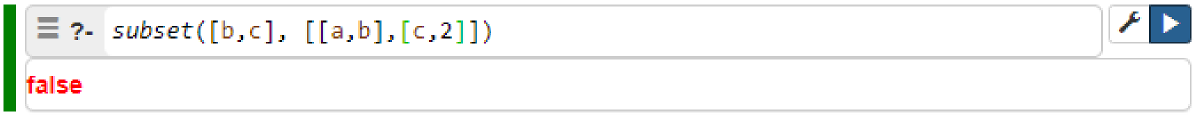
```
?- subset([a,b], [[a,b],[c,2]])
false
```

(n) {B} ⊆ G TRUE



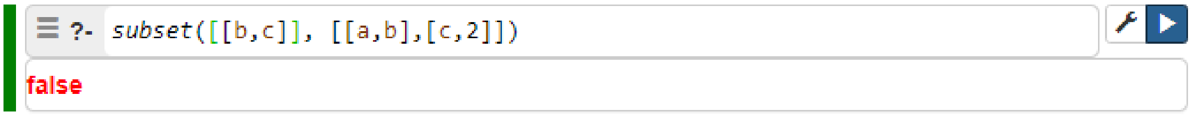
```
?- subset([[a,b]], [[a,b],[c,2]])
true
```

(o) D ⊆ G FALSE



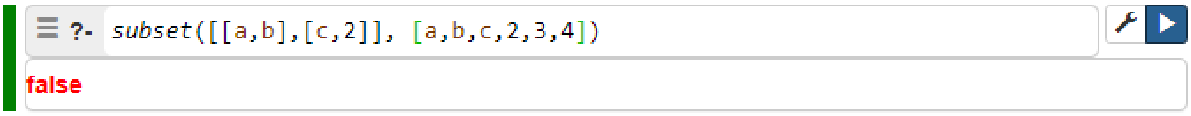
```
?- subset([b,c], [[a,b],[c,2]])
false
```

(p) {D} ⊆ G FALSE



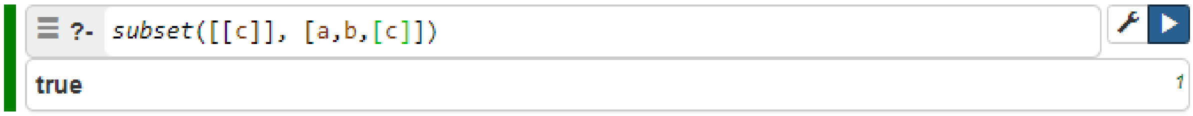
```
?- subset([[b,c]], [[a,b],[c,2]])
false
```

(q) G ⊆ A FALSE



```
?- subset([[a,b],[c,2]], [a,b,c,2,3,4])
false
```

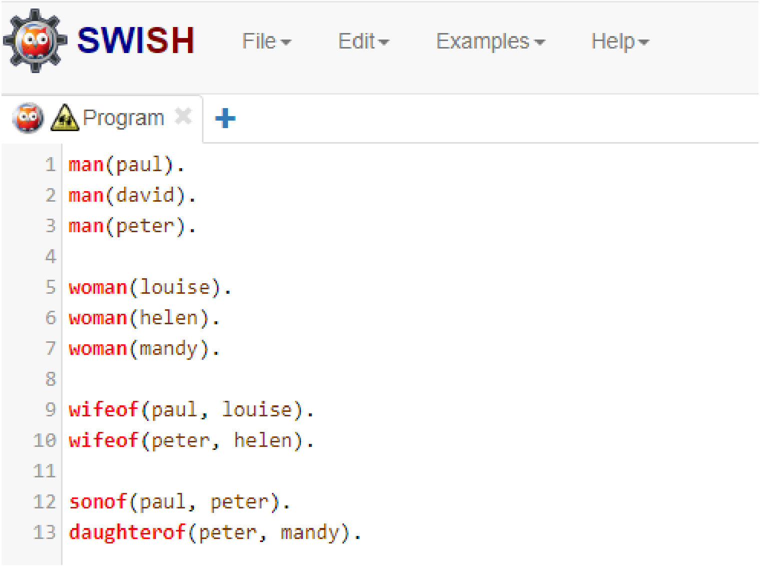
(r) {{c}} ⊆ E TRUE



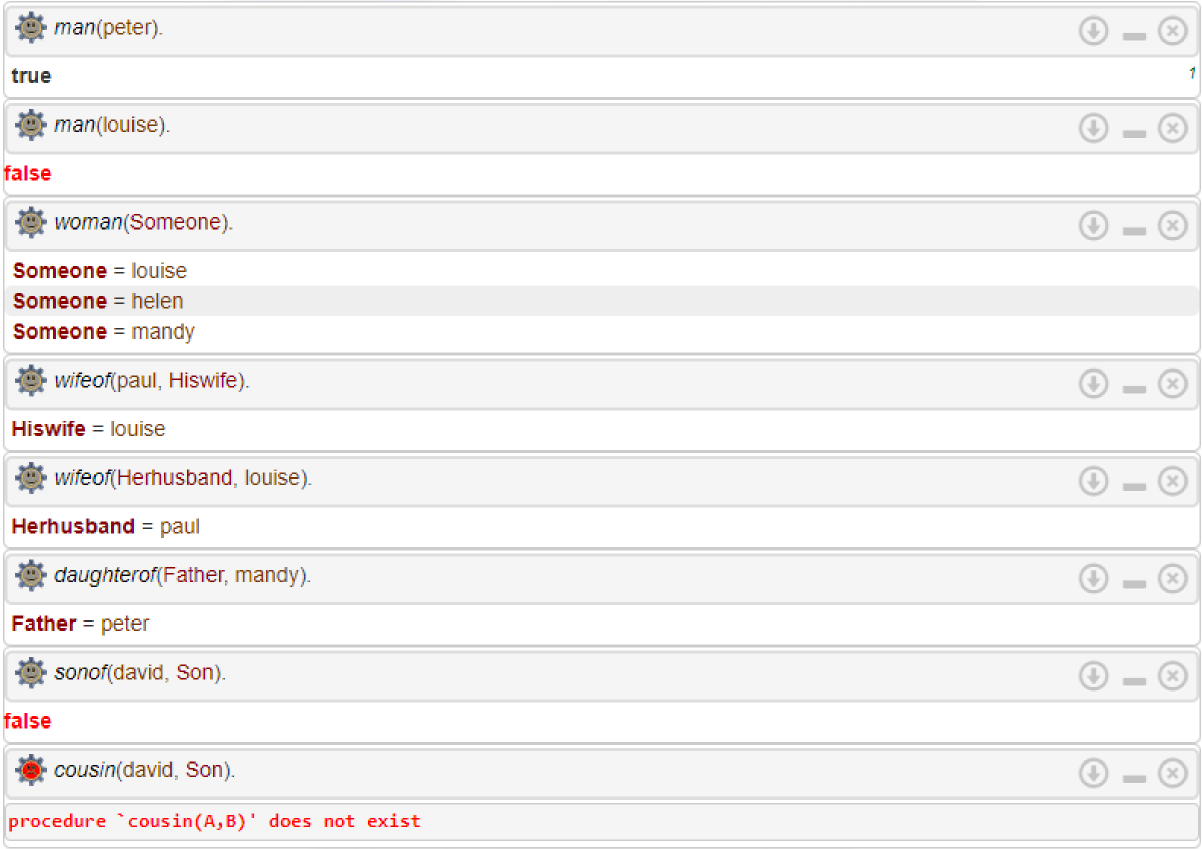
```
?- subset([[c]], [a,b,[c]])
true
```

2. Read Ritchie (2002) section 8.2 (starting on pg 12). Input the facts into the SWI-SWISH page and run the queries. To do this:

   ○ Click the + sign next to the word notebook.
   ○ Choose program.
   ○ Enter the facts (printed in the Ritchie book) into the large window.
   ○ On the right hand side of the screen you will see a smaller window, with a “?- “ at the top corner – this is the query box.
   ○ Enter your queries into the box then click the run button.
   ○ Try all the queries presented in sections 8.2 and 8.3 of the Ritchie book.



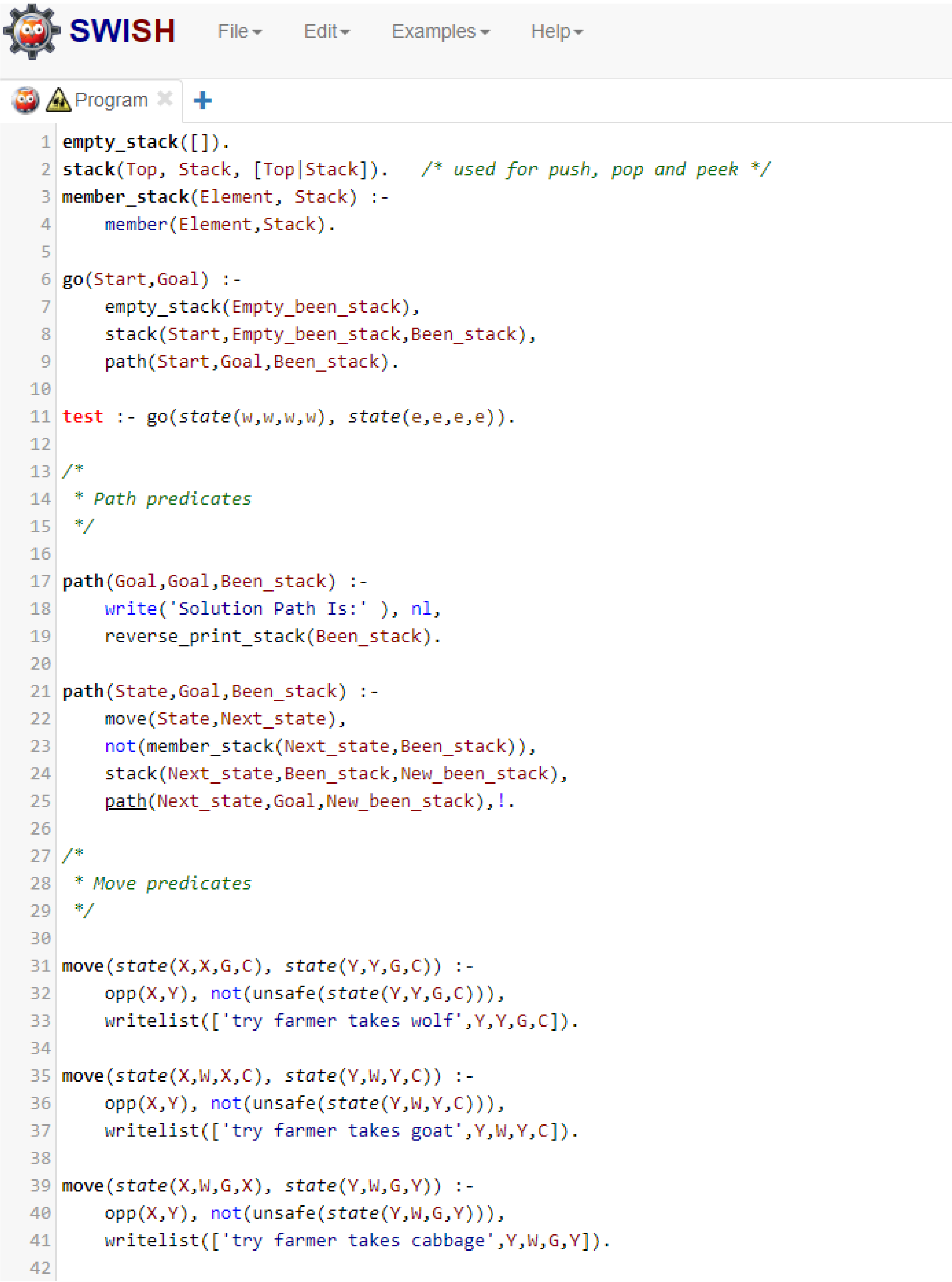
```
man(paul).
man(david).
man(peter).

woman(louise).
woman(helen).
woman(mandy).

wifeof(paul, louise).
wifeof(peter, helen).

sonof(paul, peter).
daughterof(peter, mandy).
```



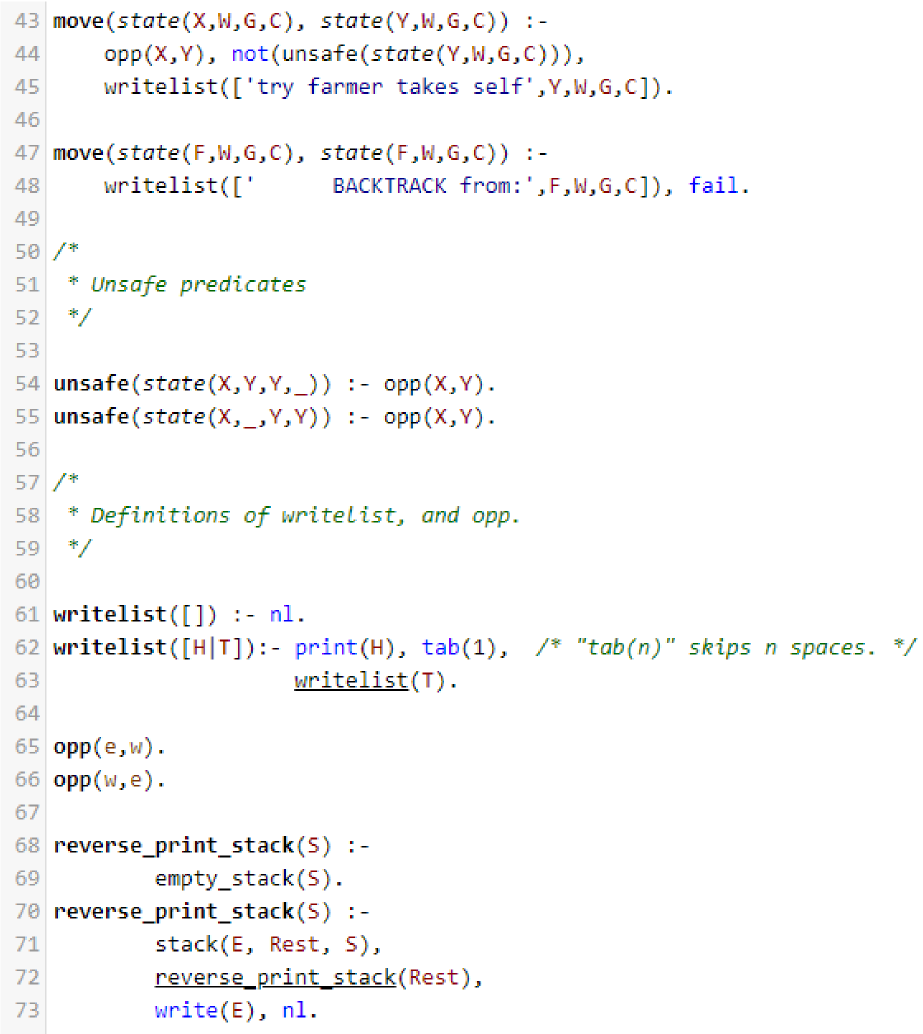
```
man(peter).
true

man(louise).
false

woman(Someone).
Someone = louise
Someone = helen
Someone = mandy

wifeof(paul, Hiswife).
Hiswife = louise

wifeof(Herhusband, louise).
Herhusband = paul

daughterof(Father, mandy).
Father = peter

sonof(david, Son).
false

cousin(david, Son).
procedure `cousin(A,B)' does not exist
```

3. Enter the Prolog version of the “crossing problem” into the SWISH program window and run it. What is the result?

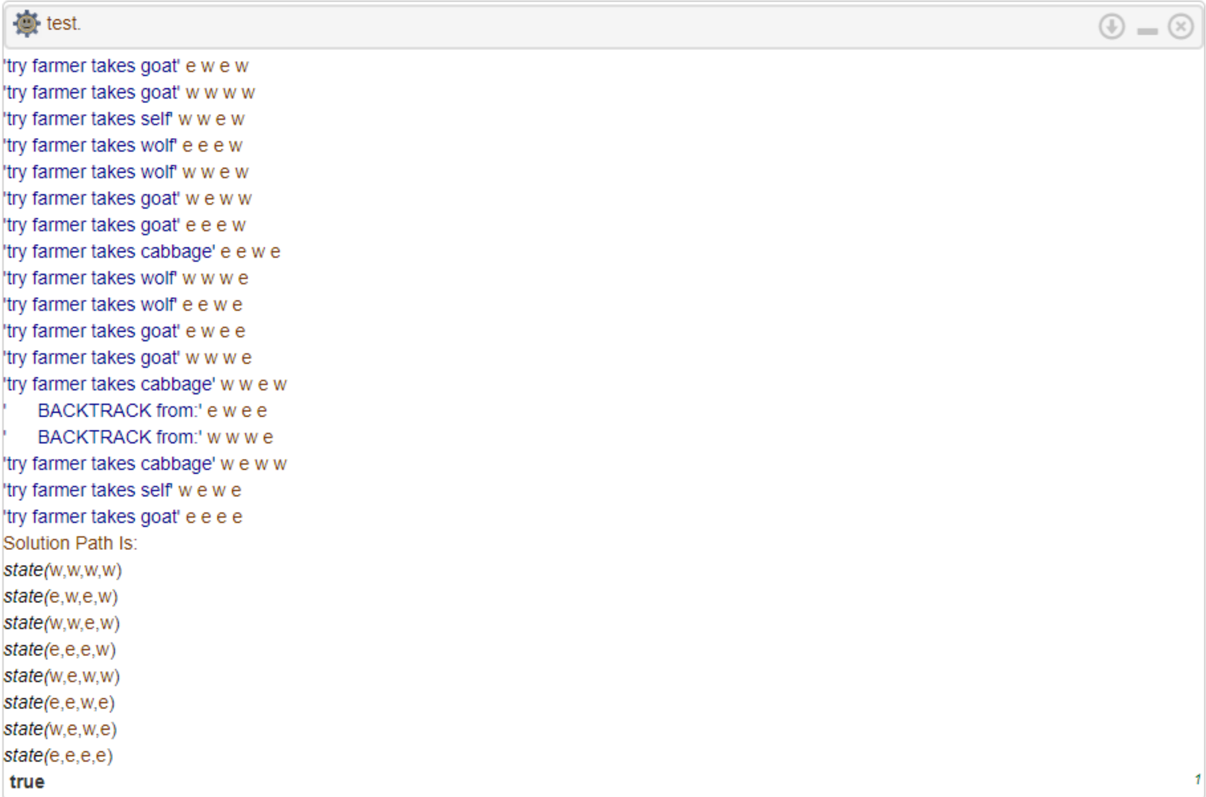
The excerpt from the lecturecast was incomplete. I was also unable to find Luger & Stubblefield (2003), however I found the code in Luger (2005), including required stack predicates earlier in the book to make it work:



```prolog
empty_stack([]).
stack(Top, Stack, [Top|Stack]).   /* used for push, pop and peek */
member_stack(Element, Stack) :-
    member(Element,Stack).

go(Start,Goal) :-
    empty_stack(Empty_been_stack),
    stack(Start,Empty_been_stack,Been_stack),
    path(Start,Goal,Been_stack).

test :- go(state(w,w,w,w), state(e,e,e,e)).

/*
 * Path predicates
 */

path(Goal,Goal,Been_stack) :-
    write('Solution Path Is:' ), nl,
    reverse_print_stack(Been_stack).

path(State,Goal,Been_stack) :-
    move(State,Next_state),
    not(member_stack(Next_state,Been_stack)),
    stack(Next_state,Been_stack,New_been_stack),
    path(Next_state,Goal,New_been_stack),!.

/*
 * Move predicates
 */

move(state(X,X,G,C), state(Y,Y,G,C)) :-
    opp(X,Y), not(unsafe(state(Y,Y,G,C))),
    writelist(['try farmer takes wolf',Y,Y,G,C]).

move(state(X,W,X,C), state(Y,W,Y,C)) :-
    opp(X,Y), not(unsafe(state(Y,W,Y,C))),
    writelist(['try farmer takes goat',Y,W,Y,C]).

move(state(X,W,G,X), state(Y,W,G,Y)) :-
    opp(X,Y), not(unsafe(state(Y,W,G,Y))),
    writelist(['try farmer takes cabbage',Y,W,G,Y]).

```



```
move(state(X,W,G,C), state(Y,W,G,C)) :-
    opp(X,Y), not(unsafe(state(Y,W,G,C))),
    writelist(['try farmer takes self',Y,W,G,C]).

move(state(F,W,G,C), state(F,W,G,C)) :-
    writelist(['      BACKTRACK from:',F,W,G,C]), fail.

/*
 * Unsafe predicates
 */

unsafe(state(X,Y,Y,_)) :- opp(X,Y).
unsafe(state(X,_,Y,Y)) :- opp(X,Y).

/*
 * Definitions of writelist, and opp.
 */

writelist([]) :- nl.
writelist([H|T]):- print(H), tab(1),  /* "tab(n)" skips n spaces. */
                   writelist(T).

opp(e,w).
opp(w,e).

reverse_print_stack(S) :-
        empty_stack(S).
reverse_print_stack(S) :-
        stack(E, Rest, S),
        reverse_print_stack(Rest),
        write(E), nl.
```



```
test.
'try farmer takes goat' e w e w
'try farmer takes goat' w w w w
'try farmer takes self' w w e w
'try farmer takes wolf' e e e w
'try farmer takes wolf' w w e w
'try farmer takes goat' w e w w
'try farmer takes goat' e e e w
'try farmer takes cabbage' e e w e
'try farmer takes wolf' w w w e
'try farmer takes wolf' e e w e
'try farmer takes goat' e w e e
'try farmer takes goat' w w w e
'try farmer takes cabbage' w w e w
'      BACKTRACK from:' e w e e
'      BACKTRACK from:' w w w e
'try farmer takes cabbage' w e w w
'try farmer takes self' w e w e
'try farmer takes goat' e e e e
Solution Path Is:
state(w,w,w,w)
state(e,w,e,w)
state(w,w,e,w)
state(e,e,e,w)
state(w,e,w,w)
state(e,e,w,e)
state(w,e,w,e)
state(e,e,e,e)
true
```

## References

Luger, W. & Stubblefield, F. (2003). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Harlow, England; Reading, Mass. Addison-Wesley.

Luger, G.F. (2005). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. 5th ed. Harlow: Addison-Wesley.