```
% :- module(birds, [multivalued/1]).
% :- use_module(nativeshell).

% BIRDS - a sample bird identification system for use with the
% Native shell.

% top_goal where Native starts the inference.

top_goal(X) :- bird(X).

order(tubenose) :-
   nostrils(external_tubular),
   live(at_sea),
   bill(hooked).
order(waterfowl) :-
   feet(webbed),
   bill(flat).
order(falconiforms) :-
   eats(meat),
   feet(curved_talons),
   bill(sharp_hooked).
order(passerformes) :-
   feet(one_long_backward_toe).

family(albatross) :-
   order(tubenose),
   size(large),
   wings(long_narrow).
family(swan) :-
   order(waterfowl),
   neck(long),
   color(white),
   flight(ponderous).
family(goose) :-
   order(waterfowl),
   size(plump),
   flight(powerful).
family(duck) :-
   order(waterfowl),
   feed(on_water_surface),
   flight(agile).
family(vulture) :-
   order(falconiforms),
   feed(scavange),
   wings(broad).
family(falcon) :-
   order(falconiforms),
   wings(long_pointed),
   head(large),
   tail(narrow_at_tip).
```

```
family(flycatcher) :-
   order(passerformes),
   bill(flat),
   eats(flying_insects).
family(swallow) :-
   order(passerformes),
   wings(long_pointed),
   tail(forked),
   bill(short).

bird(laysan_albatross) :-
   family(albatross),
   color(white).
bird(black_footed_albatross) :-
   family(albatross),
   color(dark).
bird(fulmar) :-
   order(tubenose),
   size(medium),
   flight(flap_glide).
bird(whistling_swan) :-
   family(swan),
   voice(muffled_musical_whistle).
bird(trumpeter_swan) :-
   family(swan),
   voice(loud_trumpeting).
bird(canada_goose) :-
   family(goose),
   season(winter),          % rules can be further broken down
   country(united_states),     % to include regions and migration
   head(black),                % patterns
   cheek(white).
bird(canada_goose) :-
   family(goose),
   season(summer),
   country(canada),
   head(black),
   cheek(white).
bird(snow_goose) :-
   family(goose),
   color(white).
bird(mallard) :-
   family(duck),            % different rules for male
   voice(quack),
   head(green).
bird(mallard) :-
   family(duck),            % and female
   voice(quack),
   color(mottled_brown).
bird(pintail) :-
```

```
   family(duck),
   voice(short_whistle).
bird(turkey_vulture) :-
   family(vulture),
   flight_profile(v_shaped).
bird(california_condor) :-
   family(vulture),
   flight_profile(flat).
bird(sparrow_hawk) :-
   family(falcon),
   eats(insects).
bird(peregrine_falcon) :-
   family(falcon),
   eats(birds).
bird(great_crested_flycatcher) :-
   family(flycatcher),
   tail(long_rusty).
bird(ash_throated_flycatcher) :-
   family(flycatcher),
   throat(white).
bird(barn_swallow) :-
   family(swallow),
   tail(forked).
bird(cliff_swallow) :-
   family(swallow),
   tail(square).
bird(purple_martin) :-
   family(swallow),
   color(dark).

country(united_states) :- region(new_england).
country(united_states) :- region(south_east).
country(united_states) :- region(mid_west).
country(united_states) :- region(south_west).
country(united_states) :- region(north_west).
country(united_states) :- region(mid_atlantic).

country(canada) :- province(ontario).
country(canada) :- province(quebec).
country(canada) :- province(etc).

region(new_england) :-
   state(X),
   member(X, [massachusetts, vermont, etc]).
region(south_east) :-
   state(X),
   member(X, [florida, mississippi, etc]).

region(canada) :-
   province(X),
```

```
   member(X, [ontario,quebec,etc]).

nostrils(X) :- ask(nostrils,X).
live(X) :- ask(live,X).
bill(X) :- ask(bill,X).
size(X) :- menuask(size,X,[large,plump,medium,small]).
eats(X) :- ask(eats,X).
feet(X) :- ask(feet,X).
wings(X) :- ask(wings,X).
neck(X) :- ask(neck,X).
color(X) :- ask(color,X).
flight(X) :- menuask(flight,X,[ponderous,powerful,agile,flap_glide,other]).
feed(X) :- ask(feed,X).
head(X) :- ask(head,X).
tail(X) :- menuask(tail,X,[narrow_at_tip,forked,long_rusty,square,other]).
voice(X) :- ask(voice,X).
season(X) :- menuask(season,X,[winter,summer]).
cheek(X) :- ask(cheek,X).
flight_profile(X) :- menuask(flight_profile,X,[flat,v_shaped,other]).
throat(X) :- ask(throat,X).
state(X) :- menuask(state,X,[massachusetts,vermont,florida,mississippi,etc]).
province(X) :- menuask(province,X,[ontario,quebec,etc]).

multivalued(voice).
multivalued(color).
multivalued(eats).
```

```
% Native - a simple shell for use with Prolog
% knowledge bases.  It includes expanations.

:-dynamic(known/3).

:-op(900,xfy, :).

main :-
   greeting,
   repeat,
   write('> '),
   read(X),
   do(X),
   X == quit.

greeting :-
   write('This is the native Prolog shell.'), nl,
   native_help.

do(help) :- native_help, !.
do(load) :- load_kb, !.
do(solve) :- solve, !.
do(how(Goal)) :- how(Goal), !.
do(whynot(Goal)) :- whynot(Goal), !.
do(quit).
do(X) :-
   write(X),
   write(' is not a legal command.'), nl,
   fail.

native_help :-
   write('Type help. load. solve. how(Goal). whynot(Goal). or quit.'),nl,
   write('at the prompt.'), nl.

load_kb :-
   write('Enter file name in single quotes (ex. ''birds.nkb''.): '),
   read(F),
   consult(F).

solve :-
   retractall(known),
   prove(top_goal(X),[]),
   write('The answer is '),write(X),nl.
solve :-
   write('No answer found.'),nl.

ask(Attribute,Value,_) :-
   known(yes,Attribute,Value),     % succeed if we know its true
   !.                              % and dont look any further
ask(Attribute,Value,_) :-
```

```
   known(_,Attribute,Value),       % fail if we know its false
   !, fail.

ask(Attribute,_,_) :-
   \+ multivalued(Attribute),
   known(yes,Attribute,_),         % fail if its some other value.
   !, fail.                        % the cut in clause #1 ensures
                  % this is the wrong value
ask(A,V,Hist) :-
   write(A :V),                     % if we get here, we need to ask.
   write('? (yes or no) '),
   get_user(Y,Hist),                      % get the answer
   asserta(known(Y,A,V)),          % remember it so we dont ask again.
   Y = yes.                        % succeed or fail based on answer.

% "menuask" is like ask, only it gives the user a menu to to choose
% from rather than a yes on no answer.  In this case there is no
% need to check for a negative since "menuask" ensures there will
% be some positive answer.

menuask(Attribute,Value,_,_) :-
   known(yes,Attribute,Value),     % succeed if we know
   !.
menuask(Attribute,_,_,_) :-
   known(yes,Attribute,_),         % fail if its some other value
   !, fail.

menuask(Attribute,AskValue,Menu,Hist) :-
   nl,write('What is the value for '),write(Attribute),write('?'),nl,
   display_menu(Menu),
   write('Enter the number of choice> '),
   get_user(Num,Hist),nl,
   pick_menu(Num,AnswerValue,Menu),
   asserta(known(yes,Attribute,AnswerValue)),
   AskValue = AnswerValue.         % succeed or fail based on answer

display_menu(Menu) :-
   disp_menu(1,Menu), !.             % make sure we fail on backtracking

disp_menu(_,[]).
disp_menu(N,[Item | Rest]) :-            % recursively write the head of
   write(N),write('  : '),write(Item),nl, % the list and disp_menu the tail
   NN is N + 1,
   disp_menu(NN,Rest).

pick_menu(N,Val,Menu) :-
   integer(N),                     % make sure they gave a number
   pic_menu(1,N,Val,Menu), !.      % start at one
pick_menu(Val,Val,_).             % if they didn't enter a number, use
                                   % what they entered as the value
```

```
pic_menu(_,_,none_of_the_above,[]).  % if we've exhausted the list
pic_menu(N,N, Item, [Item|_]).       % the counter matches the number
pic_menu(Ctr,N, Val, [_|Rest]) :-
   NextCtr is Ctr + 1,                % try the next one
   pic_menu(NextCtr, N, Val, Rest).

get_user(X,Hist) :-
   repeat,
   write('> '),
   read(X),
   process_ans(X,Hist), !.

process_ans(why,Hist) :-
   write_list(4,Hist), !, fail.
process_ans(_,_).

% Prolog in Prolog for explanations.
% It is a bit confusing because of the ambiguous use of the comma, both
% to separate arguments and as an infix operator between the goals of
% a clause.

prove(true,_) :- !.
prove((Goal,Rest),Hist) :-
   !,
   prov(Goal,[Goal|Hist]),
   prove(Rest,Hist).
prove(Goal,Hist) :-
   prov(Goal,[Goal|Hist]).

prov(true,_) :- !.
prov(menuask(X,Y,Z),Hist) :- menuask(X,Y,Z,Hist), !.
prov(ask(X,Y),Hist) :- ask(X,Y,Hist), !.
prov(Goal,Hist) :-
   clause(Goal,Body),
   prove(Body,Hist).

% Explanations

how(Goal) :-
   clause(Goal,Body),
   prove(Body,[]),
   write_body(4,Body).

whynot(Goal) :-
   clause(Goal,Body),
   write_line([Goal,'fails because: ']),
   explain(Body).
whynot(_).
```

```
explain(true).
explain((Head,Body)) :-
   check(Head),
   explain(Body).

check(H) :- prove(H,[]), write_line([H,succeeds]), !.
check(H) :- write_line([H,fails]), fail.

write_list(_,[]).
write_list(N,[H|T]) :-
   tab(N),write(H),nl,
   write_list(N,T).

write_body(N,(First,Rest)) :-
   tab(N),write(First),nl,
   write_body(N,Rest).
write_body(N,Last) :-
   tab(N),write(Last),nl.

write_line(L) :-
   flatten(L,LF),
   write_lin(LF).

write_lin([]) :- nl.
write_lin([H|T]) :-
   write(H), tab(1),
   write_lin(T).

flatten([],[]) :- !.
flatten([[]|T],T2) :-
   flatten(T,T2), !.
flatten([[X|Y]|T], L) :-
   flatten([X|[Y|T]],L), !.
flatten([H|T],[H|T2]) :-
   flatten(T,T2).
```

```
┌─proxzima@proxzima in ~/prologue via  v5.32.1 took 1ms [ 46%]
└─ prolog native.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
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).

?- main.
This is the native Prolog shell.
Type help. load. solve. how(Goal). whynot(Goal). or quit.
at the prompt.
> load.
|: Enter file name in single quotes (ex. 'birds.nkb'.): |: 'birds.nkb'.
> |: solve.
nostrils:external_tubular? (yes or no) > |: yes.
live:at_sea? (yes or no) > |: yes.
bill:hooked? (yes or no) > |: yes.

What is the value for size?
1  : large
2  : plump
3  : medium
4  : small
Enter the number of choice> > |: 1.

wings:long_narrow? (yes or no) > |: yes.
color:white? (yes or no) > |: no.
color:dark? (yes or no) > |: yes.
The answer is black_footed_albatross
> |:
```