Ministry of Education of Republic of Moldova
Technical University of Moldova
CIM Faculty
Anglophone Department

Report

On Logical Programming and Artificial Intelligence
Laboratory Work №4

Performed by: st. gr. FAF-131 **Gavrilita M.**
Verified by: conf. univ. dr. **Luchianov L.**

Chisinau, 2016

Laboratory Work Nr. 4

Topic:

Expert systems in Prolog.

Tasks:

- Create a expert system for a specific domain;
- Analyze the creation of an expert system based on logic (facts) and one based on rules, analyzing the backward and forward chaining.

System description:

The implemented expert system is a rule based expert system, it's domain of expertise being Solar System bodies (e.g. planets, dwarf planets, comets and asteroids). When consulting the expert system, the user is given a set of questions (multiple answers, yes or no questions), after which the expert system can decide what entry of it's knowledge base best suits the users answers. For an answer to be returned, the set of facts given by the user must coincide with one of the knowledge base entries (of course each entry must have a unique set of rules which describe it). If the list of answers that the user gave does not unify with any of the entries of the knowledge base, a reply like "No answer found." is returned.

System structure:

In figure 1 is presented the general structure of the implemented expert system as well as the main predicates defined in each part of the program. Their description can be seen below.

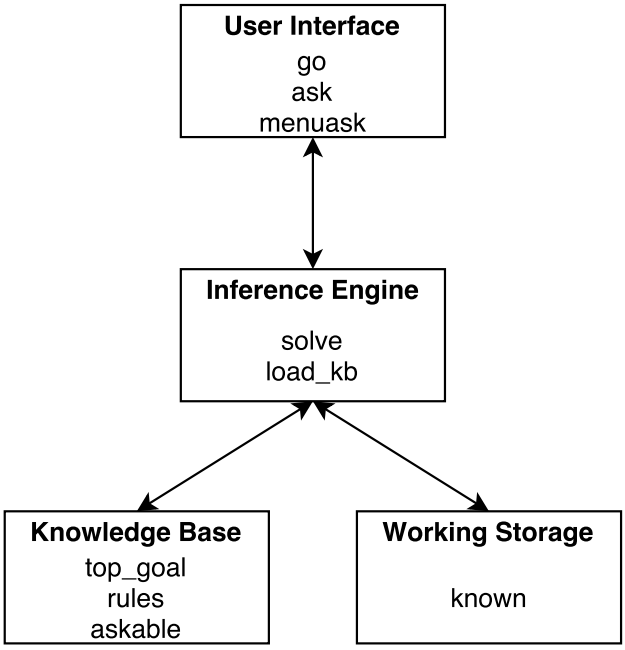

Figure 1: Structure and major predicates of the expert system

☐ User Interface - Part of the system which interacts directly with the user;

- go - Starts the execution of the user interface;
- ask - Is called when the user must provide a yes or no answer;
- menuask - Is called when the user must choose between multiple answers (e.g. choose a color).

☐ Inference Engine - Part of the system used to parse the database;

- solve - Starts the actual process of finding the answer;
- load_kb - Loads the selected knowledge base.

☐ Knowledge Base - Part of the system which stored the database entries;

- top_goal - Defines what rule will the system try to satisfy;
- rules - Set of rules comprising the data in the knowledge base;
- askable - Specifies which of the rules can be asked from the user.

☐ Working Storage - Part of the system to save the user's answers.

- known - Show all known facts provided by the user.

Code and Mentions:

Listing 1: solve predicate in the shell

```
solve :-
        abolish(known/3),
        dynamic(known/3),
        top_goal(X),
        write('The answer is '), write(X), nl.

solve :-
        write('No answer found.'), nl.
```

Here you can see the solve predicate which, when called, clears the user's answers (if any) and called the predicate top_goal. This predicate is defined at the beginning of the knowledge base. The solve predicate starts the process of finding an answer.

Listing 2: top_goal predicate in the knowledge base

```
top_goal(X) :- celestial_body(X).
```

Here is the top_goal predicate which simply permits the solver to work with any knowledge base. In this case the top_goal will seek to unify with the celestial_body predicate.

Listing 3: an entry in the knowledge base

```
celestial_body(mercury) :-
        class(planet),
        type(terestrial_planet),
        atmosphere(none),
        color(dark_grey).
```

The system will try to unify with each entry in the knowledge base, one by one. To unify, the system must satisfy all the sub-rules. To do this the system needs to know more, so, it will call the predicate ask.

Listing 4: the "askables" in the knowledge base

```
axis(X) :- ask(axis, X).
water(X) :- ask(water, X).
storm(X) :- ask(storm, X).
rings(X) :- ask(rings, X).
texture(X) :- ask(texture, X).
```

Here you can see some of the predicates being "askable". This means that in order for the system to know if it can satisfy them it must ask the user, waining for a positive or negative answer and memorizing it.

Listing 5: ask predicate in the shell

```
ask(A, V) :-
        known(yes, A, V), % succeed if true
        !.      % stop looking

ask(A, V) :-
        known(_, A, V), % fail if false
        !, fail.

ask(A, V):-
        write(A:V), % ask user
        write('? : '),
        read(Y), % get the answer
        asserta(known(Y, A, V)), % remember it
        Y == yes.       % succeed or fail
```

The ask predicate is quite simple: if the answer is already known, then check if it's positive or not and satisfy (or not) the sub-predicate you are looking at. If no answer was yet given, ask the user, remembering his answer with the built-in predicate assert.

In the end, if there is a predicate with which the system unifies, it will be returned as the answer. If there where no such predicates, the system will return "No answer found."

Listing 6: code used for the inductive use of the program

```
iknow :-
        abolish(known/3),
        dynamic(known/3),
        write('Enter your domain of interest: '),
        read(F),
        iknow_top_goal(F).

iknow_top_goal(X) :-
        asserta(known(true, tail, has)),
        asserta(known(true, tail, does_not_have)),
    ...
        asserta(known(true, texture, heart)),
        celestial_body(X).
```

```
ask(A, V) :-
        known(true, A, V),
        writeln(A:V),
        !.

menuask(A, V, _) :-
        known(true, A, V),
        writeln(A:V),
        !.
```

The parts of the code listed below are used for the inductive use of the program. The program populates the working storage with a special "answer" and calls $celestial_body(X)$. as usual. When the *ask* or *menuask* predicate want to ask about a fact, they simply print the info about the planet / comet.

Conclusion:

In this laboratory work we studied the implementation of expert systems in Prolog, as well as the rule based and fact based variations. The main working part of a rule based expert system is described in the *Code and Mentions* section.

The expert system in Prolog is using the built in functionality and backtracking to parse and find the needed solutions in a very easy to understand form, giving possibility for further development of the system and easy creation on new knowledge bases (even for a non-technical person).

The database was initially created to be used in a deductive manner: the used answered the questions proposed by the solver and after which the suitable answer was found. After some fast changes and additions, the database was also suitable for the use in an inductive manner (e.g. the user wants to know more about planet X). If to spend a bit more time, the "inductive" part could be coded more clean (i.e. to avoid repeating of code).

Appendix:

Listing 7: the shell

```
%% User Interface:

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

greeting :-
        write('This is the Native Prolog shell.'), nl,
        write('Enter load, consult, known or quit at the prompt.'), nl.

do(load) :- load_kb, !.
do(consult) :- solve, !.
do(known) :- know, !.
do(quit).

do(X) :-
        write(X),
        write('is not a legal command.'), nl,
        fail.


ask(A, V) :-
        known(yes, A, V), % succeed if true
        !.      % stop looking

ask(A, V) :-
        known(_, A, V), % fail if false
        !, fail.

ask(A, V):-
        write(A:V), % ask user
        write('? : '),
        read(Y), % get the answer
        asserta(known(Y, A, V)), % remember it
        Y == yes.       % succeed or fail


menuassert(_, _, []).
menuassert(X, A, [X|T]) :- asserta( known(yes, A, X) ), menuassert(X, A, T).
menuassert(X, A, [H|T]) :- asserta( known(no, A, H) ), menuassert(X, A, T).


menuask(A, V, _) :-
```

```
48          known(yes, A, V), % succeed if true
49          !.
50
51  menuask(A, V, _) :-
52          known(_, A, V), % fail if false
53          !, fail.
54
55  menuask(A, V, MenuList) :-
56          write('What is the value for '), write(A), write('?'), nl,
57          write(MenuList), nl,
58          read(X),
59          check_val(X, A, V, MenuList),
60          menuassert(X, A, MenuList),
61          !,
62          X == V.
63
64  check_val(X, _, _, MenuList) :-
65          member(X, MenuList), !.
66
67  check_val(X, A, V, MenuList) :-
68          write(X), write(' is not a legal value, try again.'), nl,
69          menuask(A, V, MenuList).
70
71
72
73  %% Inference Engine:
74
75  solve :-
76          abolish(known/3),
77          dynamic(known/3),
78          top_goal(X),
79          write('The answer is '), write(X), nl.
80
81  solve :-
82          write('No answer found.'), nl.
83
84  load_kb :-
85          write('Enter file name: '),
86          read(F),
87          consult(F).
88
89  know :-  known(X, Y, Z), write('> '), write(Y), write(' '), write(Z), write('
90  know.
```

Listing 8: the knowledge base

```
1  %% Knowledge Base:
2
3  top_goal(X) :- celestial_body(X).
4
5  %% Askables
```

```
6
7  tail(X) :- menuask(tail, X, [has, does_not_have]).
8  shape(X) :- menuask(shape, X, [spherical, non_spherical]).
9  surphace(X) :- menuask(surphace, X, [rocky, gas]).
10 atmosphere(X) :- menuask(atmosphere, X, [none, thin, dense]).
11 neighbourhood(X) :- menuask(neighbourhood, X, [cleared, not_cleared]).
12 color(X) :- menuask(color, X, [dark_grey, yellowish_white, blue, reddish, ora
13
14 axis(X) :- ask(axis, X).
15 water(X) :- ask(water, X).
16 storm(X) :- ask(storm, X).
17 rings(X) :- ask(rings, X).
18 texture(X) :- ask(texture, X).
19
20 %% Rules
21
22 class(planet) :-
23         shape(spherical),
24         neighbourhood(cleared).
25
26 class(dwarf_planet) :-
27         shape(spherical),
28         neighbourhood(not_cleared).
29
30
31 class(asteriod) :-
32         shape(non_spherical),
33         tail(does_not_have).
34
35 class(comet) :-
36         shape(non_spherical),
37         tail(has).
38
39
40 type(terestrial_planet) :-
41         surphace(rocky).
42
43 type(gas_giant) :-
44         surphace(gas).
45
46
47 celestial_body(mercury) :-
48         class(planet),
49         type(terestrial_planet),
50         atmosphere(none),
51         color(dark_grey).
52
53 celestial_body(venus) :-
54         class(planet),
55         type(terestrial_planet),
```

```
56          atmosphere(dense),
57          color(yellowish_white).
58
59  celestial_body(earth) :-
60          class(planet),
61          type(terestrial_planet),
62          atmosphere(dense),
63          color(blue),
64          water(yes).
65
66  celestial_body(mars) :-
67          class(planet),
68          type(terestrial_planet),
69          atmosphere(thin),
70          color(reddish).
71
72
73  celestial_body(jupiter) :-
74          class(planet),
75          type(gas_giant),
76          color(orange_white),
77          storm(great_red_spot).
78
79  celestial_body(saturn) :-
80          class(planet),
81          type(gas_giant),
82          color(pale_gold),
83          rings(yes).
84
85  celestial_body(uranus) :-
86          class(planet),
87          type(gas_giant),
88          color(blue_green),
89          axis(tilted).
90
91  celestial_body(neptune) :-
92          class(planet),
93          type(gas_giant),
94          color(pale_blue).
95
96  celestial_body(pluto) :-
97          class(dwarf_planet),
98          color(light_brown),
99          texture(heart).
100
101 celestial_body(ceres) :-
102         class(dwarf_planet),
103         color(grey).
104
105 celestial_body(a_comet) :-
```

```
106         class(comet).
107
108 celestial_body(an_asteroid) :-
109         class(asteriod).
```