AI Project Lab 2 Report
Kid's day at school

NGO Romain
N1705068D

November 2017

Contents

1 Introduction **3**

2 Launching the script **3**

3 Guessing the answer **3**

4 Defining the rules **4**

5 Asking a question **5**

6 Checking an answer **5**

7 Cleaning **5**

8 Dynamic predicate definition **6**

9 Conclusion **6**

1 Introduction

This is a report presenting the work on a Prolog script. The behavior of the script is the following: the script asks questions that can only be answered by yes or no. In the end, the script should find out what kind of activity was done by a kid based on the answers he had.

2 Launching the script

```
start :- guess(Activity),
      write('I know what you did today! Today you '),
      write(Activity),
      write('? '),
      nl,
      undo.
```

Figure 1: Starting the script

"start" is used to launch the script. What this script does is guessing an Activity first. After the activity is found, the script display the name of the activity, then jump a new line and erase the found activity with undo. To find out more about undo, see section (7) "Cleaning".

3 Guessing the answer

```
guess(played) :- played, !.
guess(drew) :- drew, !.
guess(ate) :- ate, !.
guess(learned) :- learned, !.
guess(saw) :- saw, !.
/* Whenever prolog cannot find the answer */
guess(unknown).
```

Figure 2: Guessing the answer

Whenever the answer is found by the script, we do not want prolog to backtrack and look for other solutions. hence we do not want prolog to backtrack. That is why we are using the cut(!) in this case to tell prolog to stop after finding the answer. We added a guess(unknown) to this predicate in case, prolog cannot find the solution based on the given answers.

4 Defining the rules

```
played :- numbers, check(had_fun_with_toys).
played :- check(had_fun_with_sand), !.
drew :- nature, fruit.
drew :- check(used_lots_of_colors), !.
ate :- fruit , check(ingested_meat).
ate :- check(ingested_vegetables), !.
learned :- numbers, animal.
learned :- check(tried_to_read), !.
saw :- nature, animal.
saw :- check(witnessed_human_construction), !.
```

```
numbers :- check(tried_to_count).
nature :- check(tried_to_imitate_nature).
fruit :- check(had_natural_food).
animal :- check(learned_more_about_living_beings).
```

Figure 3: Defining the rules

Here are the definitions of the predicates. Some predicates might share a same rule among them. Whenever two or more predicate share the same rule, we redefine the predicate bellow for clarity.
Sometimes, a predicate is separated on multiple lines because we do not want prolog to jump to another predicate if all the rules in the first one are not completed.
For example, for the predicate "played", the predicate is separated in two lines. If it were on one line, prolog would jump to "drew" whenever it receives a negative answer for `"check(had_fun_with_toys)"` and ignore `"check(had_fun_with_sand)"`. Which we do not want because a kid can play with numbers, have fun with sand and not toys but still played in a day.
Whenever a rule is shared among multiple predicate, we put the rule with another one. For example, numbers is shared among "played" and "learned".
If the answer to `"check(tried_to_count)"` is yes, then it is ambiguous for prolog so we must satisfy at least two conditions for prolog to be sure of the answer.
We added a cut (!) to the second line of a predicate because whenever prolog reaches this predicate, it means the answer is obvious so we do not want prolog to go back and use more resources for nothing.

5 Asking a question

```
ask(Question) :-
    write('Today, you: '),
    write(Question),
    write('? '),
    read(Response),
    nl,
    ( (Response == yes ; Response == y)
      ->
       assert(yes(Question)) ;
       assert(no(Question)), fail).
```

Figure 4: Asking a question

Asking a question is defined in the following way: prolog display the question with "write/1" then wait for an answer with "read/1". Then start the following if-then-else block. If the response is "yes" or "y" then the question is set to true and asserted into the database, else it's set to fail and asserted then it fails. When the fail happens, prolog will start to backtrack in the tree and look for other solutions. To find out more about yes/1 and no/1, see the section (6) "Checking an answer".

6 Checking an answer

```
check(S) :-
   (yes(S) -> true ; (no(S) -> fail ; ask(S))).
```

Figure 5: Starting the script

The check is based on an if-then-else block as well. If yes then set the question to true. Else another if-then-else block is launched: if no then fail else ask again. We can see that "ask/1" and "check/1" are both called which means this is a recursive loop.

7 Cleaning

This part is required if we want to start the script again without relaunching prolog. The predicate undo, removes all the assertions made by yes and no. If we don't do this, all the solutions found by prolog stay in the database.

```
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.
```

Figure 6: Cleaning the answer

8 Dynamic predicate definition

```
:- dynamic yes/1,no/1.
```

Figure 7: dynamic

In prolog, a predicate's state cannot be changed unless we set it to dynamic. Since "yes/1" and "no/1" both required the use of "assert/1" and "retract/1" which change their state, we need to set these two predicates from static to dynamic.

9 Conclusion

This mini project allowed me to find out about the logic and the reasoning behind Prolog. Since it is very different from procedural programming the kind in which I am used to. I had to change my mindset in order to understand how Prolog works.