Assignment 1

EECS 3401 A

October 2, 2020

This assignment is due on October 21 at 23:59 with no possibility of extension. Aim to complete it by the end of the reading week (October 17), so that whatever issues come up could be addressed at the office hours **before the midterm**. Late submissions will not be accepted.

> Your report for this assignment should be the result of your own **individual work**. Take care to avoid plagiarism. You may discuss the problems with other students, but do not take written notes during these discussions, and do not share your written solutions.

Question 1 [25 points]

Write and test a Prolog program `q1.prolog` that deals with family relations. Assume that the predicate `parent(X,Y)` —meaning that `X` is a parent of `Y` —has already been defined (and that the definition is loaded separately).

Your program's job is to define the following predicates:

`ancestor(X,Y)` meaning that `X` is an ancestor of `Y`, i.e. either `X` is a parent of `Y` or `X` is an ancestor of someone who is a parent of `Y`;

`common_ancestor(X,Y,Z)` meaning that `X` is a common ancestor of `Y` and `Z`, i.e. `X` is an ancestor of both `Y` and `Z`;

`closest_common_ancestor(X,Y,Z)` meaning that `X` is a closest common ancestor of `Y` and `Z`, i.e. `X` is a common ancestor of `Y` and `Z` and no child of `X` is a common ancestor of `Y` and `Z`;

`ancestorList(X,Y,L)` which holds if `X` is an ancestor of `Y` and `L` is a list of the descendants of `X` (i.e., people whose ancestor is `X`) that are ancestors of `Y` ordered from the closest to the farthest from `X`. For example, if `john` is a parent of `paul` who is a parent of `henry` who is a parent of `helen`, then `ancestorList(john,helen,L)` should succeed with `L = [paul, henry]`;

```
L = [john,
       [paul,
         [henry,
           [helen]
         ],
         [june]
       ],
       [mary,
         [adam]
       ]
     ]
```

Figure 1

`descendantTree(X,L)` which holds if `L` is a list structure representing the tree of all descendants of `X`; each node in the tree of descendants should be represented by a list whose first element is the node label and the remaining elements are the representations of its children (if any). For example, if `john`'s children are `paul` and `mary`, `paul`'s children are `henry` and `june`, `henry`'s only child is `helen`, `mary`'s only child is `adam`, and neither `adam` nor `june` nor `helen` have any children, then `descendantTree(john,L)` should succeed with a returned list as shown in Figure 1 (indentations and line breaks have been added there for readability, but your program should not do this).

You may define some auxiliary relations if that helps in defining the ones above. Test your program thoroughly before submitting it. Document your code appropriately. Do not include any `parent(X,Y)` facts in your file.

Prolog Tips and Tricks

- SWI Prolog implements "negation"[1] in the predicate `\+`, which also uses the alias `not`. This is a **unary** predicate, i.e., it takes exactly one argument. If you want to negate a "conjunction" of subgoals, you need to encase your subgoals in a set of parentheses so they appear to `not` as a single argument.

```
?- \+(subgoal1, subgoal2).      /* throws an error */
?- \+((subgoal1, subgoal2)).    /* works as intended */
?- not((subgoal1, subgoal2)).   /* same as previous */
```

- The pattern for finding an *extremum* in your knowledge base—that is, an individual which is the *smallest* or the *largest* according to some ordering relation—is to use "negation" along the lines of:

```
fattest_cat(X) :- cat(X), not((cat(Y), not(X=Y), fatter_than(Y, X)).
```

  That is, first establish the existence of an individual, then establish the non-existence of another individual which is closer to the extreme than the first.

- To find *all* individuals having a particular property *at once* (as opposed to manually cycling Prolog through different variable bindings for which the goal can be proved) can be accomplished using the built-in predicates `setof/3` and `findall/3`. Look these up in the SWI documentation.

Question 2 [20 points]

Write and test a Prolog program `q2.prolog` that solves the following puzzle:

> The police are trying to track down the gang of three kids who have been stealing pumpkins. So far, they have established the following facts: the kids' first names are Angela, Mary, and David; one is 5, one is 7, and one is 8; one has the last name Diamond, and the one with the last name Grant is 3 years older than the one with the last name Leung. You can assume Angela and Mary are female and David is male.

[1] Not strictly a logical negation—refer to the lectures and here: `https://www.swi-prolog.org/pldoc/man?predicate=%5C%2B/1`

Use the technique shown in the zebra example discussed in class (the code is available on the course web page) to find missing information on the gang: each child's age, gender, first name and last name, consistent with the data above. Encode the above data as is and do not add additional facts. Document your code appropriately.

Additionally, use your Prolog code to show whether or not the computed information uniquely identifies the culprits. Submit these test results and a short explanation of their meaning in the file `q2tests.txt`.

Question 3 [30 points]

This question is on first-order logic. Do not write Prolog code or use Prolog notation.

Consider the following information:

> Victor has been murdered, and Arthur, Bertram, and Carleton are the only suspects. **Exactly one of them is the murderer**. Arthur says that Bertram was the victim's friend, but that Carleton hated the victim. Bertram says that he was out of town the day of the murder and besides, he did not even know the guy. Carleton says that he saw Arthur and Bertram with the victim just before the murder. You may assume that everyone, except possibly the murderer, is telling the truth.

Your tasks:

(a) Write sentences in first-order logic that represent this knowledge. Also provide a **glossary** where you indicate the intended meaning of your predicate, function, and constant symbols in English.

(b) Convert the sentences into clausal form and give the resulting set of clauses.

(c) Use resolution with answer extraction to find the murderer. State how you represent the query in first-order logic and what clause (with an answer predicate) is added to the theory. Show the complete resolution derivation (in sequence or tree form), clearly indicating which literals/clauses are resolved and the unifier used.

(d) Suppose that we can no longer assume that there was only a single murderer. What sentences must you remove from the theory? Show that the modified theory no longer entails that the answer you obtained in (c) is the murderer. Do this by specifying an interpretation where the answer in (c) is not the murderer and showing that it satisfies all the axioms of the theory.

Submit your answer to this question as a PDF file `q3.pdf`.

To hand in your report for this assignment, put the files `q1.prolog`, `q2.prolog`, `q2tests.txt`, and `q3.pdf` in a directory called `a1answers` and submit it electronically by the deadline. Your Prolog code should work correctly on Prism. To submit through eClass, look for instructions on eClass. To submit from Prism, use the following command:

```
submit 3401 a1 a1answers
```