F28PL - Programming Languages

Question K (Prolog)

Tommy Lamb

H00217505

Questions begin on the next page due to size and formatting niceties.

1. Write Prolog clauses for the following scenario:
    - Jake is a person
    - Jake is a person
    - Jill is a person
    - John is a person
    - Joan is a person
    - Jake likes tomato
    - Jill likes cheese
    - Jill likes tomato
    - John likes cheese
    - Joan likes cheese
    - Joan likes tomato
    - Jake knows Jill
    - Jill knows John
    - John knows Joan
    - Joan knows Jake
    - Every person knows themself
    - If a person likes cheese and that person likes tomato then they like pizza

    ```
    person(jake).
    person(jill).
    person(john).
    person(joan).
    likes(jake, tomato).
    likes(jill, cheese).
    likes(jill, tomato).
    likes(john, cheese).
    likes(joan, tomato).
    likes(joan, cheese).
    likes(X, pizza) :- likes(X, tomato), likes(X, cheese).
    knows(jake, jill).
    knows(jill, john).
    knows(john, joan).
    knows(joan, jake).
    knows(X,X) :- person(X).
    ```

    - The only aspect of these clauses worth noting is that despite storing proper nouns, none of the names are capitalised. This is a limitation of Prolog, in that anything capitalised is considered a variable. The two-argument facts are all read as infix, for example “jake likes tomato”, “jill knows john”, “X likes pizza if X likes tomato and X likes cheese”. Variables within facts or rules can be instantiated to be anything which is both very useful and very dangerous, and can easily lead to errors. These are used to declare that anything likes pizza, provided that thing likes tomato and likes cheese.

2. Test your program by writing questions to check:
    - Whether Jake likes pizza
    - Which persons like pizza
    - Which persons know other persons who likes pizza

    - `likes(jake, pizza).`
    - `likes(X, pizza).`
    - `knows(X,Y), likes(Y,pizza), X\=Y.`

    ❖ Behaving exactly as expected, Prolog returns all the possible instantiations of X such that the query holds true; it also correctly returns that Jake does not like pizza. The only item of note is the stipulation that X cannot unify with Y in the third query – "`X\=Y`". This is required otherwise Prolog will also return people who just like pizza, and not necessarily "know *other* persons who likes pizza". The reason for such a result is the rule "every person knows themself". Thus a person - Joan - knows herself, and Joan likes pizza, so Joan knows someone (Joan herself) who likes pizza. With the stipulation in place, X and Y cannot both be instantiated to Joan (as an example).

3. Write Prolog clauses for the following scenario:
    - Aberdeen is a place
    - Dundee is a place
    - Edinburgh is a place
    - Glasgow is a place
    - Kirkcaldy is a place
    - St Andrews is a place
    - There are 60 miles between Aberdeen and Dundee
    - There are 60 miles between Dundee and Edinburgh
    - There are 45 miles between St Andrews and Edinburgh
    - There are 10 miles between Dundee and St Andrews
    - There are 60 miles between Dundee and Aberdeen
    - There are 30 miles between St Andrews and Kirkcaldy
    - There are 35 miles between Kirkcaldy and Edinburgh
    - There are 45 miles between Glasgow and Edinburgh
    - The miles from a first place to a second place is the miles between the first place and the second place or the miles between the second place and the first place
    - The distance between a first and second place is the miles from the first to the second or the miles from the first to a third place plus the distance between the third and second place

    ```
    o place(aberdeen).
    o place(dundee).
    o place(edinburgh).
    o place(glasgow).
    o place(kirkcaldy).
    o place(standrews).
    o distance(aberdeen, dundee, 60).
    o distance(dundee, edinburgh, 60).
    o distance(standrews, edinburgh, 60).
    o distance(dundee, standrews, 10).
    o % distance(dundee, aberdeen,60). %Superfluous, serves only to give duplicates
    o distance(standrews, kirkcaldy, 30).
    o distance(kirkcaldy, edinburgh, 35).
    o distance(glasgow, edinburgh, 45).
    o distance(X,X,0). %Added to enable recursivedistance to work

    o finddistance(X,Y,Z) :- distance(X,Y,Z), place(X),place(Y) ; distance(Y,X,Z),
      place(X), place(Y).

    o indirectdistance(X,Y,A,Z) :- finddistance(X,Y,Z) ; finddistance(X,A,B),
      finddistance(A,Y,C), Z is C+B.

    o recursivedistance(From, To, Distance) :- finddistance(From, To, Distance),! ;
         recursivedistance(From, X, D1) , recursivedistance(To, Y, D2),
         recursivedistance(X, Y, D3), Distance is D1+D2+D3,!.
    ```

    ❖ The three place fact `distance(X,Y,Z)` is read as "the distance between X and Y is Z".
    The rule `finddistance/3` was deliberately named so to avoid the infinite looping that would result from the intuitive reusing of the name `distance`. The penultimate scenario would intuitively be written as `distance(X,Y,Z) :- distance(Y,X,Z).` though this would loop until the stack memory was exhausted.

    The final scenario is encoded in `indirectdistance/4`, where A is the third location, and works as logically expected. However it requires that there be a third place to which the first and second locations directly link, rather than being some arbitrary pass-through location. An improvement on

this is realised in `recursivedistance/3`. In this rule two pass through locations are used, X and Y, to eliminate the problem of finding an exact midpoint location between the stipulated From and To (or X and Y from `indirectdistance`). Acting recursively this function will find go through an arbitrary number of arbitrary locations in attempt to satisfy that they can all be connected to the specified destinations – and thus the two locations themselves be connected. The downside to this recursion this that should such a connection be impossible it will run until the stack is exhausted. This is reasonable as for any possible connection there are an infinite number of paths that could be taken between them (with revisiting). Furthermore, in Prolog a computational failure of this nature is tantamount to a logical failure, so this 'crash' is a perfectly acceptable result for an impossible query. Despite searching for two intermediary locations, since they can both be instantiated to be the same place the rule succeeds even when only one intermediary is required. It is this which requires the inclusion of the fact `distance(X,X,0).` as it allows the base case (`finddistance(X,Y,Z)`) to succeed when attempting to connect two variables instantiated to the same location. The duplicate entry commented out using % also caused the recursive program to occasionally enter an infinite loop, for reasons not entirely clear. Its removal rectified the issue, though remains as a comment as it was specified by the question. This recursive rule does not however find the most efficient route, rather the route depends entirely on the ordering of data in the Knowledge Base – another issue with how Prolog works, though theoretically negateable.

The use of the cut operator `!` is to force Prolog to only give a single result, otherwise an infinite number of results would be provided, as there exists an infinite number of paths between two places (given any single path exists) should places be revisited.

4. <u>Test your program by writing questions to find distances between:</u>
    - Edinburgh and St Andrews
    - Aberdeen and Glasgow

    - `finddistance(edinburgh, standrews, Distance).`
    - `indirectdistance(aberdeen, glasgow, Via, Distance).`
    - `recursivedistance(aberdeen, glasgow, Distance).`

    ❖ The first query returns the expected result of `Distance = 60`, however this is followed by `false`. The reason for this is thought to be down to the use of logical or. While one clause of the rule returns a result, the other does not and thus `false` is also returned as a valid result.

    The second query fails outright because there is no Via location which directly connects both Aberdeen and Glasgow (see comments on `indirectdistance` and `recursivedistance` above). The third query succeeds however, going via Edinburgh and Dundee.