Lógica y Metodos avanzados de Razonamiento - Exercises

18 de Diciembre 2006

Two more exercises for Resolution:

1. Write down the following sentences in First-Order-Logic:

   > Lisa and Karl are persons. Lisa likes persons who have green pets. Canary birds are pets. Among the Canary birds of Karl there are some green ones.

   Use the predicates $person/1$, $pet/1$, $canary/1$, $likes/2$, $owns/2$, $hasColor/2$ with the following intuitive meanings:

$person(x)$	... "x is a person."	p3cml
$pet(x)$	... "x is a pet."	
$canary(x)$	... "x is a Canary bird."	
$likes(x, y)$	... "x likes y."	
$owns(x, y)$	... "x owns y."	
$hasColor/2$	... "x has color y."	

   Transform the First-Order Logic formulae to clausal form and proof by SLD-refutation that Karl likes lisa, i.e. proof the goal:

   $$\leftarrow likes(lisa, karl)$$

2. The following set of facts defines a train schedule for Austrian cities using the predicate
   `train(CityStart,CityEnd,TimeStart,TimeEnd)`
   which describes direct trains with their departure and arrival times:

   ```
   train(vienna, innbruck, 1430, 1800).
   train(innsbruck, bregenz, 1905, 2100).
   train(innsbruck, feldkirch, 1830, 2015).
   train(feldkirch, bregenz, 2030, 2055).
   train(vienna, linz, 0800, 1000).
   train(linz, innsbruck, 1300, 1815).
   ```

   The following clauses extend the facts from above by the predicate
   `tc(CityStart,CityEnd,TimeStart,TimeEnd)`
   which defines the (transitive) train connections[1]:

   ```
   tc(A,C,TimeA,TimeC) ← tc(A,B,TimeA,TimeB), tc(B,C,TimeB1,TimeC).
   tc(X,Y,TimeX,TimeY) ← train(X,Y,TimeX,TimeY).
   ```

   (a) Write down two SLD-refutations representing two different answer substitutions for the query:

   $$\leftarrow \texttt{tc}(\texttt{vienna}, \texttt{bregenz}, \texttt{Start}, \texttt{End})$$

   (b) The program presented above does not work in PROLOG, due to non-termination problems. rewrite the program, such that it works in PROLOG.

[1] Like in PROLOG, variables are written in upper case letters

3. We defined a predicate `add/3` in the last lecture using the constant 0 and the successor function `s/1`, cf. file `natNumbers.pl` on the lecture web page. In the example file you find analogous definitions of predicates `mult/3` and `exp/3` with the intuitive meanings:

   `mult(X,Y,Z)` ...X*Y=Z

   `exp(X,Y,Z)` ...X^Y=Z

   e.g. for

   ```
   ?- mult(s(s(0)),s(s(s(0))),X).
   ```

   the programm answers:

   ```
   yes
   X = s(s(s(s(s(0))))))
   ```

   or analogously for:

   ```
   ?- exp(s(s(0)),s(s(s(0))),X).
   ```

   the programm answers:

   ```
   yes
   X = s(s(s(s(s(s(s(0)))))))
   ```

   Run the program and try out several queries with these predicates. Formulate a query cpmputing the the result of

   ```
   s(s(0)) * ( s(s(0)) ^ s(s(s(0)))
   ```

   i.e., $2 * 2^3$.

   Next, formulate the same query using integers and prolog's built-in arithmetic.

4. Take a look at the file `emperors.pl`. Define a predicate `sibling/2` which computes all siblings, i.e. persons with common parents. Similarly, generalize this to a predicate `same_generation/2` which determines persons at the same level of the genealogy. For avoiding the inference of answers like "X is a sibling of X", you can use the infix pre-defined predicate '\==/2' in the body of a rule.

5. Using PROLOG's list notation and arithmetic, write a predicate `scalar/3` which computes the scalar product of two vectors, i.e.

$$(x_1, \ldots, x_n) * (y_1, \ldots, y_n) = \sum_{i=1}^{n} x_i * y_i$$

   e.g.:

   ```
   ?- scalar([2,4,6],[6,3,-1],X).
   X=18
   ```

6. Redefine the predicates '`functor/3`' and '`arg/3`' using '`=..`', i.e. implement two predicates '`myfunctor/3`' and '`myarg/3`' which show exactly the same behavior as the built-in predicates.

7. Write a predicate `transp(X,Y)` which computes the transposed matrix, for a given list of lists. E.g.

```
  [[1,2,3,4],                Y=[[1,5,9],
   [5,6,7,8],      ==>          [2,6,0],
   [9,0,a,b]]                   [3,7,a],
                                [4,8,b]]
```

i.e., on the query:

```
?- transp([[1,2,3,4],[5,6,7,8],[9,0,a,b],Y).
```

your program should answer:

```
yes
Y=[[1,5,9],[2,6,0],[3,7,a],[4,8,b]]
```

You may assume that all lists have the same length (i.e., that the input is indeed a matrix)

SWI-Prolog is available for download for various platforms (including documentation) at: `http://www.swi-prolog.org/`

Note that solving these exercises is for your benefit! You can send solutions and questions to me by Monday January 8th via e-mail: *axel@polleres.net*