Logic and Functional Programming

Labworks 10

April 26, 2021

1 Review questions

1. Consider the following program

```
father(john,bill).
mother(ann,bill).
parent(X,Y):-father(X,Y).
parent(X,Y):-mother(X,Y).
```

   (a) How many facts and how many rules are in this program?

   (b) Indicate the function symbols, the predicate symbols, and the variables used in this program.

   (c) What predicates are defined in this program?

2. Use the Martelli-Montanari algorithm to compute a most general unifier of the terms

   (a) `f(X,Y,Z)` and `f(a,Z,h(a))`

   (b) `f(g(X),g(c),Y)` and `f(g(g(Y)),X,a)`

   (c) `f(h(b),X,X,Y)` and `f(h(b),g(Y),g(g(Z)),g(a))`

3. Consider the problem of arranging three 1's, three 2's, ..., three 9's in sequence so that for all $1 \leq i \leq 9$ there are exactly i numbers between successive occurrences of i. Use Prolog to define the relation `niceList(L)` for lists which have this property.

   Suggestion. Note that `L` is such a nice list if it has 27 elements and the following property: for all $1 \leq i \leq i$, `L` contains a sublist of the form

$$[i, \underbrace{\_,\ldots,\_}_{i \text{ times}}, i, \underbrace{\_,\ldots,\_}_{i \text{ times}}, i]$$

   By default, SWI Prolog displays only the first 10 elements of long lists. You can change this setting by evaluating the query

```
?- set_prolog_flag(answer_write_options,
      [quoted(true),portray(false),max_depth(28),spacing(next_argument)]).
```

2 Homework (Deadline: May 11, 2021)

1. Consider the Prolog representation of sets described in Lecture 10. Define the following relations on sets:

   (a) `union(A,B,C)` which holds if `C` is the union of sets `A`, `B`.

   (b) `diff(A,B,C)` which holds if `C` is the set difference of sets `A`, `B`.

   (c) `included(A,B)` which holds if A is subset of B.

2. Consider natural numbers represented in the Peano arithmetic

   ```
   nat ::= 0 | s(nat)
   ```

   and define the predicate `pow(A,B,C)` which takes as input parameters the numbers A, B represented in Peano arithmetic, and instantiates the output parameter `C` with the number `A` to power `B`, represented in Peano arithmetic. For example,

   ```
   ?-  pow(s(s(0)),0,C).        % compute 2^0 = 1
   C = s(0).
   ?- pow(s(s(0)),s(s(0)),C).   % compute 2^2 = 4
   C = s(s(s(s(0)))).
   ```

3. In Prolog we can not define functions, but we can define predicates that behave like functions. For example,

   - Instead of the function that computes $\min(x, y)$, we can define the predicate `minim(X,Y,R)` which binds `R` to $\min($`X`$,$ `Y`$)$:

     ```
     minim(X,Y,X) :- X =< Y.
     minim(X,Y,Y) :- X > Y.
     ```

   - For the function $f : \mathbb{R} \to \mathbb{R}$, $f(x) = \begin{cases} -x & \text{if } x < 0, \\ x^2 & \text{if } 0 \leq x < 2, \\ x/2 & \text{if } 2 \leq x \leq 10, \\ 7 & \text{if } x > 10 \end{cases}$

     we can define the relation `fx(X,R)` to hold if `R` is bound to the value of $f($`X`$)$:

     ```
     fx(X,R) :- X<0,R is -X.
     fx(X,R) :- X>=0,X<2,R is X*X.
     fx(X,R) :- X>=2,X=<10,R is X/2.
     fx(X,7) :- X>10.
     ```

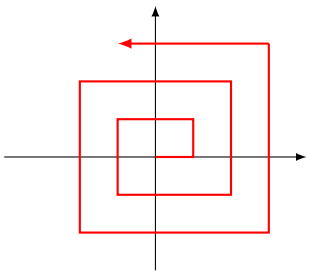
   Define the following relations:

   (a) `gcd(A,B,D)` which holds if `D` is the greatest common divisor of `A` and `B`. It is assumed that `A` and `B` are nonnegative integers. Note that

- If `B` $= 0$, the gcd is `A`.
- If `A` $<$ `B`, the gcd of `A` and `B` coincides with the gcd of `B` and `A`.
- Otherwise, the gcd of `A` and `B` coincides with the gcd of `B` and `R`, where `R` is the remainder of dividing `A` by `B`.

(b) `nextPair((X,Y),(X1,Y1))` which takes as input a pair of integers `(X,Y)` that represent the coordinates of a point in the Cartesian plane, and returns the coordinates `(X1,Y1)` of the next point on the spiral depicted below:

$(0,0),(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1),\ldots$



Note that the pair next to (x, y) is

- $(x+1, y)$ if $y \leq 0$ and $y \leq x \leq -y$,
- $(x, y+1)$ if $x > 0$ and $-x < y < x$,
- $(x-1, y)$ if $y > 0$ and $-y < x \leq y$,
- $(x, y-1)$ if $x < 0$ and $x < y \leq -x$.

4. Define the relation `subList(L1,L2)` to hold if the elements of list `L1` occur in list `L2` in the same order. For example:

```
?- subList([2,4,6],[1,2,3,4,5,6,7]).
true .
?- subList([4,2,6],[1,2,3,4,5,6]).
false .
```

5. Define the relation `countElems(Lst,[X,Y],N)` which takes as inputs a list `Lst` and a list of two elements `[X,Y]`, and instantiates `N` with the number of occurrences of `X` and `Y` in `Lst`. For example,

```
?-countElems([a,d,c,a,z,c,b,b,c],[a,c],N).
N = 5.
```

6. Define the predicate `shift_left(Lst1,Lst2)` which holds if list `Lst2` is a left rotational shift of list `Lst1`. For example,

```
?- shift_left([1,2,3,4,5],L1),shift_left(L1,L2).
L1=[2,3,4,5,1],
L2=[3,4,5,1,2].
```

7. Define the predicate `shift_right(Lst1,Lst2)` which holds if list `Lst2` is a right rotational shift of list `Lst1`. For example,

```
?- shift_right([1,2,3,4,5],L1),shift_right(L1,L2).
L1=[5,1,2,3,4],
L2=[4,5,1,2,3].
```

8. Define the following relations using one or more accumulators:

   (a) `fact(N,R)` which takes as input a positive integer `N` and instantiates the output parameter `R` with the value of `N!`. For example,

   ```
   ?- fact(5,R).
   R = 120.
   ```

   (b) `fib(N,R)` takes as input a positive integer `N` and instantiates the output parameter `R` with the value of the `N`-th Fibonacci number. For example

   ```
   ?- fib(1,R).      ?- fib(2,R).      ?- fib(7,R).
   R = 1.            R = 1.            R = 13.
   ```

9. Define the predicate `isSorted(Lst)` which takes as input a list of numbers and returns `true` if the elements of `Lst` are in increasing order. For example,

```
?- isSorted([1,2,3]).  ?- isSorted([]).  ?- isSorted([2,1,3]).
true.                  true.             false.
```

10. Consider the relation `elim(Lst,E,Rest)` which has input parameter a list `Lst`, and output parameters `E` and `Rest`. The following program clauses define this relation to hold when `R` is an element of `Lst` and `Rest` is the result of removing `E` from `Lst`:

```
elim(E,[E|Rest],Rest).
elim(E,[X|Lst],[X|Rest]) :- elim(E,Lst,Rest).
```

For example,

```
?- elim([1,2,3],E,Rest).
R=1,
Rest=[2,3];
R=2,
Rest=[1,3];
R=3,
Rest=[1,2];
false.
```

A permutation of a list `Lst` is a rearrangement of the elements of `Lst`. Note that `Perm` is a permutation of a non-empty list `Lst` if `Perm` is of the form `[E|Rest]` where `E` is an element of `Lst` and `Rest` is a permutation of `Lst` without `E`.

(a) Use the previous observation to define the relation `perm(Lst,Perm)` to hold if `Perm` is a permutation of list `Lst`.

(b) Use the previously defined predicates `isSorted` and `perm` to define the relation `sortList(Lst,S)` which takes as input a list of numbers `Lst` and binds the output parameter `S` to the sorted version of list `Lst`. For example,

```
?- sortList([],S).      ?- sortList([5,1,3,2,4],S).
S= [].                  S = [1,2,3,4,5].
```

11. Define the relation `insert(E,Lst,R)` which takes as inputs a number `E` and a sorted list of numbers `Lst`, and binds `R` to the sorted list produced by inserting `E` in `Lst` at the right place. For example,

```
?- insert(4,[],R).      ?- insert(4,[1,2,8]).
R = [4].                R= [1,2,4,8].
```

12. Define the relation `insertionSort(Lst,Sorted)` which binds the output parameter `Sorted` to the sorted version of the list of number `Lst`, by implementing the algorithm of sorting by insertion. In Prolog this algorithm is defined as follows:

    - If `Lst` is a nonempty list, take its head `H`, sort its tail `T`, and insert `H` in the sorted tail at the right place.

13. Another way to sort a nonempty list of numbers is the following:

    1) Take its head `H` and tail `T`,
    2) Split the tail `T` into
       `T1` = the elements of tail smaller than `H`, and
       `T2` = the elements of tail larger than `H`,
    3) Sort (recursively) the lists `T1` and `T2`, and
    4) Concatenate the sorted version of `T1` with `[H|S2]`, where `S2` is the sorted version of list `T2`.

       The predefined predicate `append(L1,L2,R)` can be used to bind `R` to the result of concatenating lists `L1` and `L2`.

    Use this method to implement the relation `sortV2(Lst,S)` which binds the output parameter `S` to the sorted list of numbers `Lst`.

14. Define a predicate `twoTimesLonger(L1,L2)` which holds if list `L2` is two times longer than list `L1`. For example,

```
?- twoTimesLonger([],[]).
true.
?- twoTimesLonger([1,2],[a,b,c,d]).
true.
?- twoTimesLonger([a],[a,b,c]).
false.
```

15. Define `sum_and_squareSum(Lst,S1,S2)` which takes as input a list of numbers `Lst` and instantiates

    `S1` with the sum of elements in `Lst`, and
    `S2` with the sum of squares of elements in `Lst`.

    For example,

```
?- sum_and_squareSum([],S1,S2).
S1 = 0,
S2 = 0.
?- sum_and_squareSum([1,-3,2 ,0],S1,S2).
S1 = 0,
S2 = 14.
?- sum_and_squareSum([1,2,3,4],S1,S2).
S1 = 10,
S2 = 30.
```

16. Define `isPalindrome(Lst)` which holds if and only if `Lst` is a list which coincides with its reverse. For example,

```
?- isPalindrome([]).
true.

?- isPalindrome([a,b,b,c]).
true
```

17. Define the predicate `rgb(Lst)` if and only if `Lst` is a list of 0 or more `r`, followed by zero or more `g`, followed by 0 or more `b`.

    For example,

```
?- rgb([]).
true.
?- rgb([b,r,g]).
false

?- rgb([r,g,g,b]).
true.

?- rgb([b,b]).
true.
```