Programming Paradigms Third session about logic programming

Hans Hüttel

8 December 2020

Problem 1

The natural numbers were defined in the podcast as

```
nat(zero).
nat(succ(X)) :- nat(X).
```

Implement the following relations on natural numbers: −, · and minimum. Use the definitions of addition and ≤ from the podcast for today.

```
nat(zero).

nat(succ(X)) :- nat(X).

leq(zero, Y) :- nat(Y).

leq(succ(X), succ(Y)) :- leq(X, Y), nat(X), nat(Y).

add(X, zero, X) :- nat(X).

add(zero, Y, Y) :- nat(Y).

add(succ(X), Y, succ(R)) :- add(X, Y, R), nat(X), nat(Y), nat(R).

sub(X,zero,X) :- nat(X).

sub(succ(X),succ(Y),zero) :- Y = X, nat(X).

sub(succ(X),succ(Y),zero) :- Y = X, nat(X).

sub(succ(X),Y,W) :- W = succ(Z), sub(X,Y,Z), nat(Z).

mult(X,zero,zero) :- nat(X).

mult(X,succ(Y),Z) :- mult(X,Y,V), nat(V), add(V,X,Z), nat(Z).

min(X,Y,X) :- nat(X),nat(Y),leq(X,Y).

min(X,Y,Y) :- nat(X),nat(Y),leq(Y,X).
```

• Describe how Prolog enables computation of subtraction from addition.

An alternative solution is sub1(X,Y,Z) := add(Y,Z,X).This says that X - Y = Z if Y + Z = X. To find N1 - N2, simply make the query sub1(N1-N2,R)The R returned is the value of N1 - N2.

Use the representations from the solution to the previous problem to formulate and test Prolog queries that determine if the following equations have a solution:

- x = 1 + 2
- x + 2 = 3
- $x \cdot x + 1 = 5$
- $x \leq \min(x, y)$

where x and y are natural numbers.

```
solve1(X) :- nat(X), mult(X,succ(X),succ(succ(succ(succ(succ(zero)))))).
solve2(X) :- nat(X), add(X,succ(succ(zero)),succ(succ(succ(succ(zero)))).
solve3(X) :- nat(X), mult(X,X,V), succ(V) = succ(succ(succ(succ(succ(zero))))).
solve4(X,Y) :- nat(X), nat(Y), nat(Z), min(X,Y,Z), leq(X,Z).
```

Implement the Fibonacci function as a Prolog predicate fib.

```
fib(0,1).
fib(1,1).
fib(X,Z) :- N1 is X-1, N2 is X-2, fib(N1,Z1), fib(N2,Z2), Z is Z1+Z2.
```

Problem 4

Implement Prolog predicates prefix(xs, ys) and suffix(xs, ys) that tell us if the list xs is a prefix or suffix of ys.

```
prefix([],_).
prefix([X|XS],[X|YS]) :- prefix(XS,YS).
suffix([],_).
suffix([X|XS],[X|XS]).
suffix(XS,[_|YS]) :- suffix(XS,YS).
```

Implement the Prolog predicate double(xs, ys) that tells us that the list ys duplicates every element in the list xs. As an example, we should have that double([1, 2, 3], [1, 1, 2, 2, 3, 3]).

double([],[]). double([X|XS],[X|[X|XXS]]) :- double(XS,XXS).

Problem 6

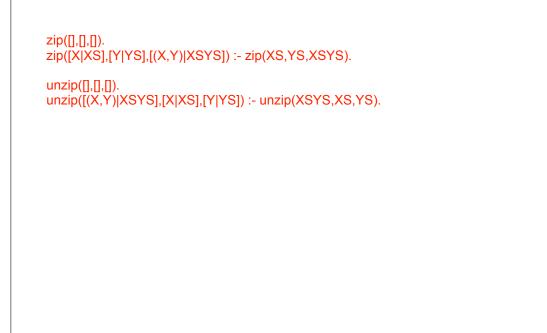
Implement zip(xs, ys, zs) to compute the pairing of the elements of the lists xs and ys. Then, implement unzip(xs, rs, ss) for the reverse.

As an example, we should have that

 $\mathtt{zip}([1,2,3],[3,4,5],[(1,3),(2,4),(3,5)])$

and that

unzip([(1,3),(2,4),(3,5)],[1,2,3],[3,4,5])



Implement prefix and suffix in terms of append.

append([],YS,YS). append([X|XS],YS,[X|XSYS]) :- append(XS,YS,XSYS).

prefix1(X,Y) :- append(X,_,Y).
suffix1(X,Y) :- append(_,X,Y).

The last two definitions should remind us that existential quantification is our friend in Prolog.

X is a prefix of Y if there exists some list that, when appended to X, gives us Y. X is a suffix of Y if there exists some list that, when X is appended to it, gives us Y.

For the miniproject

We can describe a directed graph with weighted edges by 3-place predicate edge.

Below is an example of this can be done. edge(a,b,3) tells us that there is an edge from vertex a to vertex b with weight 3.

edge(a,b,3). edge(a,c,5). edge(b,d,4). edge(b,a,2). edge(c,d,4).

Write a predicate leastinpath(X,Y,V) that holds if V is the least weight found in any path from X to Y. One way of approaching this is to find the list of weights that appears in any path from X to Y, but do we really need that?