Logic week3

Michelle Bergin

October 22, 2018

1 Practical session in class

If this is submitted as a zip, any .pl's are added in the zip

2 Ch3 execises 3.1-3.5

3.1 In the text, we discussed the predicate

```
descend(X,Y) :- child(X,Y).
descend(X,Y) :- child(X,Z),
                descend(Z,Y).
```

Suppose we reformulated this predicate as follows:

```
descend(X,Y) :- child(X,Y).
descend(X,Y) :- descend(X,Z),
                descend(Z,Y).
```

Would this be problematic?

Yes, if I am understanding how prolog does recursion this would go on for until it ran out of stack space. As to why I can't quite articulate. I think the way prolog works is that it needs a base case in the recursive call to GET to the base case to stop. So this would continually just call it's self. But maybe I am wrong...

3.2 First, write a knowledge base using the predicate directlyIn/2 which encodes which doll is directly contained in which other doll. Then, define a recursive predicate in/2 , that tells us which doll is (directly or indirectly) contained in which other dolls. For example, the query in(katarina,natasha) should evaluate to true, while in(olga, katarina) should fail.

```
directlyIn(irina,natasha).
directlyIn(natasha,olga).
directlyIn(olga,katarina).
in(X,Y):- directlyIn(X,Y).
```

```
in(X,Y):- directlyIn(X,Z), in(Z,Y).
```

3.3 We have the following knowledge base:

directTrain(saarbruecken,dudweiler).

directTrain(forbach,saarbruecken).

directTrain(freyming,forbach).

directTrain(stAvold,freyming).

directTrain(fahlquemont,stAvold).

directTrain(metz,fahlquemont).

directTrain(nancy,metz).

That is, this knowledge base holds factsn about towns it is possible to travel between by taking a direct train. But of course, we can travel further by chaining together direct train journeys. Write a recursive predicate travelFromTo/2 that tells us when we can travel by train between two towns. For example, when given the query

travelFromTo(nancy,saarbruecken).

it should reply yes.

3.4 3.4 Define a predicate greater_than/2 that takes two numerals in the notation that we introduced in the text (that is, 0, succ(0), succ(succ(0)), and so on) as arguments and decides whether the first one is greater than the second one.

greater_than(succ(_),0).

greater_than(succ(X),succ(Y)):- greater_than(X,Y).

3.5 3.5 Binary trees are trees where all internal nodes have exactly two children. The smallest binary trees consist of only one leaf node. We will represent leaf nodes as leaf(Label) . For instance, leaf(3) and leaf(7) are leaf nodes, and therefore small binary trees. Given two binary trees B1 and B2 we can combine them into one binary tree using the functor tree/2 as follows: tree(B1,B2) . So, from the leaves leaf(1) and leaf(2) we can build the binary tree tree(leaf(1),leaf(2)) . And from the binary trees tree(leaf(1),leaf(2)) and leaf(4) we can build the binary tree tree(tree(leaf(1), leaf(2)),leaf(4)) .

Now, define a predicate swap/2 , which produces the mirror image of the binary tree that is its first argument. For example:

?- swap(tree(tree(leaf(1), leaf(2)), leaf(4)),T).

T = tree(leaf(4), tree(leaf(2), leaf(1))).

yes

OK here we go

Base case is hitting a leaf.

otherwise go left or right

```
swap(tree(leaf(X),leaf(Y)),tree(leaf(Y),leaf(X))).
swap(tree(X,leaf(Y)),Z):- swap
swap(tree(leaf(X),Y),tree(Y,leaf(X))).
swap(tree(X,Y),tree(Y,X)).
```

3 Practical in-class Week 3

- Implement multiplication recursively. Ie mul(5,6) = 5+mul(5,5).
  mul(_,0,0).
  mul(X,Y,Z):- Y1 is Y - 1, mul(X,Y1,Z1), !, Z is X + Z1.
- Implement the factorial function using recursion and the integers. In order to do this you will have to do some "arithmetic". Since = is about unification, in order to do arithmetic you won't use =! Instead assignment is done using the keyword is:
  V is X+1
  factorial(1,1). factorial(X,Z):- X1 is X - 1, factorial(X1,Z1),!, Z is X * Z1.
- Since you now have a sum function in prolog (from the previous week), write:
  - multiplication
    mulNum(X,succ(0),X).
    mulNum(X,succ(Y),Z):- mulNum(X,Y,Z1), !, add(X,Z1,Z).
  - greater than
    greater_than(succ(_),0).
    greater_than(succ(X),succ(Y)):- greater_than(X,Y).
  - less than
    less_than(0,succ(_)).
    less_than(succ(X),succ(Y)):- less_than(X,Y).
  - equals
    equal(X,Y):- X = Y.
  - ge
    ge(X,Y):- equal(X,Y); greater_than(X,Y).
  - le
    le(X,Y):- equal(X,Y); less_than(X,Y).

- ne

  ne(X,Y):- not(equal(X,Y)).

- Once you have multiplication for numerals, write the factorial function with numerals instead of ints.

  add(0,Z,Z).

  add(succ(X),Y,Z):- add(X,succ(Y),Z).

  mulNum(X,succ(0),X).

  mulNum(X,succ(Y),Z):- mulNum(X,Y,Z1), !, add(X,Z1,Z).

  factNum(succ(0),succ(0)).

  factNum(succ(X),Z):- factNum(X,Z1), !, mulNum(Z1,succ(X),Z).

- Implement fibonacci with ints.
- Implement fibonacci with numerals.
- write a function to determine if a number is prime. Use the mod operator mod(X,Y) is a function that returns the remainder of X when divided by Y.