Homework 7 — Simple programs in Prolog — assigned Wednesday 9 April, due Wednesday 16 April

Total number of points available on this homework is 140. Full credit is equivalent to 100 points.

Reading assignment

Read Chapter 3 of *Clause and Effect*.

Read the paper: Jacques Cohen, "Describing Prolog By Its Intepretation And Compilation", Communications of the ACM, 28(12), 1311–1324, 1985.

Optional reading: Chapters 1, 2, and 3 of *Programming in Prolog*.

7.1 (49pts)

Complete Worksheets 6, 7, 8, 9, 11, 13, and 15 from *Clause and Effect* (7pts each).

7.2 Arithmetic evaluator (20pts)

The relation `ev` is between an expression in a very restricted arithmetic language described below, and a number which that expression evaluates to, in this little language. You can assume that the first argument is instantiated.

```
?- ev(3,N).
N=3

?- ev([+,3,4], N).
N=7

?- ev([+,3,[*,8,0.25]], N).
N=5
```

Expressions in the "little language" are defined as

```
E ::= number
    | [+, E, E]
    | [*, E, E]
    | [exp, E]
    | [sin, E]
    | [cos, E]
```

These have the obvious numeric meanings when evaluated. Write the rules for `ev`.

7.3 Tic-tac-toe (25pts)

Write a Prolog program that allows a human user to play tic-tac-toe against the computer. Either the computer or the human can move first. The computer may choose its moves using any policy whatsoever, including random. The first player to move is the X player, the second is the O player.

Suggested representations of a tic-tac-toe board include these possibilities: `[x,o,o,b,x,b,b,b,x]` or `[[x,o,o],[b,x,b],[b,b,x]]`. (`b` stands for an unoccupied square.) Both of these examples are lists intended to represent a winning board for X in which X has conquered the bend (the diagonal from top left to bottom right).

You do not have to worry about interactive input-output. It suffices to write a predicate `computersmove(L1,L2)`, which, when given an instantiated board L1, produces a unique board L2 as the outcome of the computer's move. Assume that the human will repeatedly query this predicate.

Hints: Consider writing a predicate `xwins(L)`, meaning that the board L is a winning board for X, and a similar predicate `owins(L)`. Consider writing a predicate `okmove(L1,L2)`, meaning that from the board L1 it is legal to get to the board L2 in the very next move.

7.4 Smarter tic-tac-toe (6pts)

Same as above, but the computer should use a policy that guarantees it will not lose if it plays first.

7.5 [ML] Adequacy (20pts)

Use the signature `DIGRAPH` and structure `Digraph` for directed graphs you developed in Exercise 5.2 to implement Prim's minimum spanning tree algorithm (see Exercise 3.1).

The input to the algorithm is a directed graph, and the output is a tree. Trees are graphs. The edges of the tree should be directed away from the root.

You need to write a function `minSpanTree` within a structure `Prim`, which matches a signature for minimum spanning tree computation:

```
signature MST =
   sig
      type graph
      val minSpanTree: graph -> graph
   end

structure Prim: MST =
   struct
      ...
   end
```

Are the directed graph signature and structure adequate to this task? Would you make any changes in retrospect? Would you make changes to `MST`? Discuss.

7.6 [ML] Adequacy (20pts)

Suppose you are given the compiled code of the SML structure `Poly :> POLY`, but not its source form, so you are only able to *use* it, and that only through the signature `POLY`. How difficult is it to write a function to do polynomial division?

How to turn in

Turn in your code by running

~roshan/handin your-file

on a regular UNM CS machine.

You should use whatever filename is appropriate in place of your-file. You can put multiple files on the command line, or even directories. Directories will have their entire contents handed in, so please be sure to clean out any cruft.