COMP 356
Programming Language Structures
Class 26: Additional Prolog notes

The main topic for today's class is "Deficiencies of Prolog." For this topic, please refer to section 16.7 of the textbook. These notes provide some background on two small separate topics, which may help with the homework.

## 1 Comma notation in list heads

We already discussed the meaning of Prolog fragments like `[A|B]`. This notation can be extended using commas to match multiple items at the head of the list. For example, `[A,B|C]` matches a list whose first two elements are `A`, `B`, with the remainder of the list being `C`.

## 2 More sophisticated data structures

Prolog is capable of representing sophisticated data structures. As a simple example[1] of this, the code given below can be used to represent, manipulate, and query a sorted binary tree that stores a single integer data value at each node. The basic idea is to define a relation `node(L, D, R)`, where `L` represents the left child of the node (which could be the special value `empty`), `R` represents the right child of the tree, and `D` represents the data value stored at the node.

```
/* relation to check whether some value occurs in the tree */
isin(K, node(_, K, _)).
isin(K, node(L, D, R)) :- K < D, isin(K, L).
isin(K, node(L, D, R)) :- K > D, isin(K, R).

/* adding a node at the proper position in the tree with no attempt at
   balancing the tree */
insert(K, empty, node(empty, K, empty)).
insert(K, node(L, D, R), node(L2, D, R)) :- K < D, insert(K, L, L2).
insert(K, node(L, D, R), node(L, D, R2)) :- K > D, insert(K, R, R2).

/* do an inorder traversal of the tree, accumulating the node values in
   a list */
inorder(empty, []).
inorder(node(L, D, R), Z) :- inorder(L, LL), inorder(R, RL),
                             append(LL, [D], Z1),
                             append(Z1,  RL, Z).
```

Please see the accompanying file `tree.pl` to experiment with these definitions.

[1] This example is based closely on Professor Wahls' lecture notes.