**Comparing Functional Programming and Logic Programming**

*Name: Stephen Griffin*

*ID: 18482934*

Haskell, being a functional programming language, and Prolog, being a logical programming language have many features in common, although they have many important differences as well. Both languages focus heavily on the use of recursion and it is necessary in the code used for both programs. The binary trees are defined through recursion as Nil, and an empty tree, or a node with an element and two branches of the same tree type. In Haskell the Tree is defined using the recursive Data Type ‘Bintree’ , while the lack of type in Prolog allows us to use the tuple to represent a tree as the atom nil or the tuple tree (Left, X, Right), where both Left and Right are trees under this definition and X is any element.

Given that the definition in both cases is recursive, the insertion, search and orders are also done this way. Insertion is done considering the tree a Binary search tree, where you add an element on the left branch of a tree if it’s smaller than the element in the current node, and on the right branch of the tree if it’s bigger. Both in Prolog and in Haskell the same logic follows. Given the strong typing in Haskell, it’s necessary to add to the typing of the function that the elements of the Tree show be of a type that’s an instance of Ord, that way we can compare them.

For the search the same idea is used. If the element is in the current node, you’ve found it. If it isn’t you should proceed searching the left tree if the element is smaller and the right tree if the element being searched is bigger. The orders in binary trees are also done using recursion. The inorder list of elements of a binary tree returns the list of elements for the left branch, the element in the current node and then the elements on the right branch. If you use preorder you should take the element in the node, the ones on the left tree and the ones on the right tree and if you use postorder you should take first the left and right branch and at last the element in the current node.

The implementation of all of these functions does not differ heavily between one language and the other, both using the recursion allowed by the language to solve the given task.

The big thing that differentiates one programming paradigm from the other is not needed to solve these tasks. Prolog is known for its use of backtracking to solve problems. If a predicate fails it searches other options until it finds a valid one or there are no options left. Haskell and functional languages in general have the feature of higher order functions, where you can pass a function as an argument and use it to simplify code. This also wasn’t necessary to solve the problem at hand.

At last, if you wanted to add a remove operation on a binary you also would have a similar result in both languages. It would be needed to search for an occurence of the element on the tree. If found, it would be necessary to replace the node with this element with the top element of one of its branches, and then remove that element from the branch used. If both branche are nil then the node should be replaced with a nil pointer.