CA341

*Comparative Programming Languages*

***Student:*** *Nigel Guven*

***ID:*** *14493422*

Formal Logic Programming v Functional Programming

Analysis Report

**Due Date:** *10am 11th December*

1. **Assignment Objective**

Implement the following program in a ***logic programming language*** and a ***functional programming language*** wherein the program returns the ***longest common prefix*** of a ***list of strings*** of ***variable length***. For example, given the list

[“interview”, “interrupt”,”integrate”, ”intermediate”]

the longest common prefix is “inte”.

1. **Table Of Contents**

**Logic Paradigm**

**Functional Paradigm**

**Prolog – Logic Style Language//**

**Haskell – Functional Style Language//**

**Prolog Implementation//**

**Prolog Exercise Analysis//**

**Haskell Implementation//**

**Haskell Exercise Analysis//**

**Analytical Comparison//**

**Glossary//**

**References//**

1. **Logic Paradigm**

**1.1 Overview**

Logic Programming is a type of programming paradigm which is primarily based upon formal logic. It is based upon predicate calculus where the program is based upon a set of facts which can be seen as either statements or relationships which are of course true. There are then axioms and clauses which act like functions and can either be true or false. Logical Languages are high-level which require translation into machine language before they can be executed. There are three main concepts in logical programming: deduction; negations; non-procedural programming. Examples of logic languages are Prolog, F-Logic and QL.

**1.2 Deductions**

Computation is made by logical deduction. Data is expressed by statements which are either true or false. First-Order Logic is the construct for the design of the language which allows for deduction to occur. Given the statement ‘Charles Babbage invented the first mechanical computer’, through deduction we can create the clauses, ‘there exists X such that X is Charles Babbage and X invented the first mechanical computer’. Deductions are axioms that may infer interpretative statements upon a supplied set of facts.

**1.3 Negations**

Negation by Failure is an inference rule in logic programming. It is used to derive a statement given that all possible clauses have failed. NAF is primarily used to achieve completeness in a logical language where for every axiom in a program, the axiom may produce a demonstrable value. This is also known as being semantically valid.

**1.4 Non-Procedural Programming**

Logic Programming Languages incorporate the traits of the non-procedural paradigm. This paradigm specifies what needs to be computed but not how it should be done. The programmer must specify the set of items involved in computation, the relationships that some of them may have and the constraints which must hold to some Boolean value for the problem to be solved. The compiler then decides how to satisfy the constraints that are made. Another name for this paradigm is the declarative paradigm.

1. **Functional Paradigm**

**2.1 Overview**

Functional Programming is a paradigm which treats the computation of functions as mathematical expressions. The order of evaluation is dependent upon how the expression is declared. The programmer must understand what is to be computed rather than how it should be computed. Functional programs do not contain any variables; ergo data is immutable and circumvented of changing states. The expressions in this paradigm also have no side-effects which output anything other than the actual value.

The removal of variables in functional programming comes with its own problems. Iterative constructs like loops are non-existent which means the only way to implement iteration is through recursive functions. Examples of functional programming are Haskell, Wolfram and Lisp.

**2.2 Higher Order Functions**

Higher-Order Functions take other functions as arguments or can return a function as its output. They are essential in functional programming languages because they allow for more useful concepts like recursion and induction to take place with the program.

**2.3 Immutable Values**

Immutable values are common in functional programming. They do not allow an object to be modified from its created state. This makes the data more secure. In functional programming, you would compose functions to obtain a value that is required. The value cannot be saved and changed later. Rather than focussing on designing the data, there is an incentive for examining what a function can do.

**2.4 Lazy Evaluation**

Functional languages evaluate by needs. They do not have constructs for calling by value/reference. If the function argument is evaluated the value is stored for some purpose. Often referred to as lazy evaluation, it avoids the need to repeat evaluations which can reduce the running time of a program. This strategy for evaluation only fails when combined with imperative features such as exception handling and I/O because the order in which they operate becomes indeterminate.

**2.5 Referential Transparency**

When an expression is used we are only interested in what it can output as a value. Functional programs do not have assignment statements which allow for variables to change. An expression is declared to be referentially transparent if it can be replaced with the value it should output and not change the overall behaviour of the program. These expressions are known as pure functions and are a key concept of the functional paradigm.

1. **Prolog – Logic Style Language**

**Haskell – Functional Style Language**

**Prolog Implementation**

**Prolog Exercise Analysis**

**Haskell Implementation**

**Haskell Exercise Analysis**

**Analytical Comparison**

**Glossary**

**References**

<https://www.csee.umbc.edu/courses/331/fall12/notes/prolog/prologoverview.ppt.pdf>

<https://thebittheories.com/levels-of-programming-languages-b6a38a68c0f2>

<http://wiki.c2.com/?LogicProgramming>

<http://www.computing.dcu.ie/~davids/CA208_Prolog_2p.pdf>

<http://cs.union.edu/~striegnk/learn-prolog-now/html/node90.html>

<https://en.wikipedia.org/wiki/Functional_programming#Use_in_industry>

<https://medium.com/@cscalfani/so-you-want-to-be-a-functional-programmer-part-1-1f15e387e536>

<https://maryrosecook.com/blog/post/a-practical-introduction-to-functional-programming>

<http://hkupty.github.io/2016/Functional-Programming-Concepts-Idioms-and-Philosophy/>

<http://www.computing.dcu.ie/~hamilton/teaching/CA320/notes.html>