DCU School Of Computing
CASE3
CA341 Comparative Programming Languages
Assignment 2

'Comparing Functional and Logic Programming Languages by Differentiaiting Their Performance With a Binary Search Algorithm'

November 2021

Student Name: Katie Clancy
Student No: 19452724

Table Of Contents

1. An Introduction

Programming languages under the umbrellas of both Logical and Functional programming operate suggestively of their paradigm[1] classification names; Logical programming is based around facts and rules and how we represent them, while Functional programming uses functions in a mathematical style.

Chosen Logic Programming Language:

- Prolog

Chosen Functional Programming Language:

- Haskell

There are differences in both appearance and operation between the languages, but the most noticeable that distinguishes them from one another is arguably their 'building Blocks'.[2] when programming in Haskell, functions are the basis of the code, while when programming in Prolog, predicates take their place as the basis.

Aside from their many differences, both langauges fall under the paradigm of being *declarative*; they define their task and desired outcome.[3]

2. An Insight Into The Languages

Functional programming, known for its error handling, is based on building new functions [4]. What this entails is that pure functions are used, which have no shared state and mutable variables are not used, however new ones can be returned. A language that falls under the functional paradigm would inhabit characteristics like the use of recursion and conditional expressions in moulding their code, meaning you won't see loops implemented. While Object Oriented languages like Python are focused mainly on using expressions to successfully execute a process, Haskell is focused on the result of execution [4].
In comparison with a language like C or Java, a Haskell program that implelemts a similar algorithm striving for the same results will look far shorter and compact than that of a C program. This allows it to be maintained in a much easier fashion.
Haskell as a language is general purpose and *statically typed,* meaning that the checking of constraint types is carried out at compile time as opposed to run time [5].
The most distinguishable feature of Haskell is that its programs are written as mathematical functions.
Haskell is a desirable language when working with large datasets or high security requirements. This is because it offers 'code safety', by means of using *lazy evaluation;* where blocks of code are only evaluated when they are needed. (This also means their runtime is lower). Memory leaks and overflows are completely avoidable and scarce due to their use of automatic memory management. The simple fact that the code itself appears organized and compact means managing and debugging will be a more efficient task. This also offers benefits in scalability and helps ease *refactoring* (a process by which you can change your code without it having an effect on its external behaviour, an easy way to avoid technical debt).[4]

Logical programming in basic terms represents the display of knowledge. In logical programming, *interference* is used to manipulate said knowledge. Definitively, it relays how to reach a goal, not what the goal should be.
It uses predicate calculus in its complexion, which leads it to its predicate structure; if I want to state that the dog eats his dinner, in Python for example, I would write something like this:

dog.eat(dinner)

After my function definition.
But in Prolog, it would be written as follows:

eats(dog, dinner).

Predicates are used as the main definitive aspect of logic programming languages like Prolog.
Unlike Haskell, Prolog is *not statically typed*. While this means that it loses out on the benefits this provides, it holds its own with a range of unique characteristics.
By nature, Prolog is *nondeterministic*. What this means is that there can be more than a single answer to a predicate statement, however it can also succeed with just one defined. It also entails that sometimes on the same input, the program's behaviour may differ. This is because the program's path of execution isnt completely decided by the defined computation [6].
Type inference, a characteristic inhabited by Prolog, is the ability to know the type of a variable that sits inside a query when the query succeeds; we assign a type to the variable in simpler terms. While this may increase development time slightly, it harbours many benefits, including error probability rate reduction and easy-to-read code (which in combination counteract the extra time taken in development).
As Prolog is based around the display of knowledge, its built in list handling features come in very handy for users along with its recursion abilities.

3. An Analysis of Structure And Performance

Both programs have a similarity in structure at first glance. They both begin by defining the tree.
In the Haskell implementation, recursion is used in each function block to insert a new node into the tree, search it for a supplied node value, and in each instance of traversal. It is also used in the Prolog implementation at each new action.
The use of recursion in each means they both contain a *Base Case*; in the logical implementation for example, the base case is that N is the node or value in the tree:

search(N tree(N, Left, Right)).

The scripts (as mentioned before) are similar in appearance; tidy, ordered and mostly readable without the necessity for comments to explain the code, although they should be included anyway.

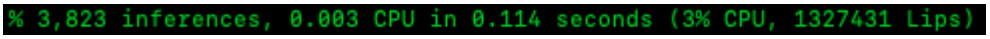
However Prolog is more concise, making it slightly less readable. Although compact code can be great for error correcting and maintainability, you have to work harder to understand the code, especially in the absence of comments. Haskell displays its operations in a clearer fashion, making it superior in my opinion in terms of readability.

Worst case time complexity for binary search trees in general is O(n). When it comes to performance, good goal and clause ordering contributes greatly to its efficiency. This is obvious and unproblematic in most cases when implementing, but can create some tricky errors if misunderstood.

The runtime of each program is displayed as follows:

Haskell

Prolog



Overall, both languages handled the task (at a low level of complexity) quite well. Binary Search was not majorly difficult to implement in either, however I did stumble across some silly errors in Prolog that took me longer than they should have to correct due to its inferior readability.
However, I have some experience programming in both languages. I suspect for a beginner programmer, both languages would be tricky to navigate. I even found it meticulous to remember its nature having not practiced it in some time. With Prolog, it takes far longer to familiarise yourself with its stubborn syntax and ordering in comparison to Haskell.

4. Conclusion

In line with my findings, while each language stands for many different benefits, the superior one for me would have to be Haskell. It's a lot less problematic, far easier to interpret and just overall a much easier language to get to know.
My impression of Prolog both from this assignment and my time studying it as a module, is that it is inflexible, and easier to utilize as a programming language than an actual problem solver. It follows a depth first search strategy that in my opinion makes it harder to implement.

There is more to be sought in terms of testing these languages against each other with perhaps a more complex problem that would recover some starker differences between them, but for now, my only comparison between the two are their characteristics.

5. *References / Sources:*

1. *'What is a programming paradigm?' - https://en.wikipedia.org/wiki/Programming_paradigm*
2. *'Logical v Functional' - https://stackoverflow.com/questions/8297574/difference-between-logic-programming-and-functional-programming*
3. *'Declarative programming' - https://searchitoperations.techtarget.com/definition/declarative-programming*
4. *'Why Haskell - an intro to functional proramming' - https://medium.com/geekculture/why-haskell-a9117c42da12*
5. *'Statically typed explained' - https://stackoverflow.com/questions/1517582/what-is-the-difference-between-statically-typed-and-dynamically-typed-languages*
6. *'Nondeterminism in programming' - https://slikts.github.io/concurrency-glossary/?id=nondeterministic-vs-deterministic*
7. *'What is code refactoring?' - https://www.bmc.com/blogs/code-refactoring-explained/*
8. *'The difference between logical and functional programming' - https://www.geeksforgeeks.org/difference-between-functional-and-logical-programming/*