Analytical Comparison

Introduction

I have implemented this program to find the longest common prefix of a list of strings in two different programming languages. The first programming language I used is Haskell. This is a purely functional programming language which means its function rarely have any side effects. Haskell is used mainly for rapid application development. The second programming language I have used is Prolog. This is mostly a general-purpose logic programming language. It is mainly used for artificial intelligence and computational linguistics. The reason I have chosen these two languages is that I am required to write the program in a logic programming language and functional programming language. I have chosen these ones over other logic and functional programming languages as I am very familiar with both.

Description

In the Haskell program I have included two functions which include the lcpHelper and the lcp function. The lcpHelper function is used to find the longest common prefix of two strings. This function is then used by the lcpHelper function to compare adjacent strings in the list, the result of the two strings which have been compared will replace the two strings. This is repeated until all strings in the list have been compared. In the prolog program I have three predicates which are lcp, lcpList and lcpOfList. The lcp predicate finds the longest common prefix of two strings. The lcpList predicate recursies through the list and finds the longest common prefix of adjacent strings and then replaces them with their longest common prefix. The lcpOfList predicate calls the lcpList function and then converts the result it receives, which will be character code(s) into a string which can be read by humans.

Similarities

In the Haskell implementation I have included a function which will find the longest common prefix of two strings and I've also included a function which will use this function to find the longest common prefix of a list of strings. Similarly, in prolog I use a predicate which finds the longest common prefix of two strings and another predicate which uses that predicate to find the longest common prefix of a list of strings. In both implementations I use a top down approach, this means I use a pyramid of functions/predicates that are called and pass down their result down through the code until a result has been returned. The use of base cases can be seen in both implementations, these are used to tell the program when to stop its recursion, without these base cases the programs would have no idea of when to stop its recursion and would result in a stack memory error. In both programs there are base cases for the lcp function/predicate, these base cases will be stop the function/predicate once the end of one of the strings being compared has been reached. The use of recursion is present in both programs. I use recursion for the same purpose in both programs, to find the longest common prefix of two strings. Both programs recurse through both strings until it reaches the end of one string or the head of either strings don't match each other. The reason I have used recursion in both implementations is I am very familiar with it and it is a very efficient way to compare two strings. The method in which I find the longest common prefix of two strings is very similar as well, I split the string into a head and a tail, then I compare the heads of each them with the equals operator, then the tail is passed back to the function/predicate until they don't match anymore or one of the strings is empty.

Differences

The most significant difference between the Prolog implementation and the Haskell implementation is the use of predicates over functions. In the Haskell implementation I use two functions to find the longest common prefix of a list of strings. Functions are called with parameters and return a

result based on them parameters. Whereas in the Prolog implementation I use three predicates to find the longest common prefix of a list of strings. In Prolog functions don't exist, it only has predicates which are a description of a relationship between terms. They will only evaluate to true or false, but we can trick them into acting like functions by giving them an unknown variable which will return a true result. The Prolog contains more code as I found it impractical to require the user to call the function lcpList with 2 unknown variables as they only wanted to receive the string as their result rather than the string and the character code(s). In the Haskell implementation there is no need for this as it will return the string without the need to convert anything or assign any unknown variables to the result of the function. Another difference between the two implementations is the way in which I use the lcp function/predicate to compare all adjacent elements of the list given. In the Prolog implementation I do this using manual recursion, I find the longest common prefix of the two strings and then replace them with their longest common prefix. In the Haskell I simple use a built-in function called foldl, this function takes the second argument and the first item of the list which in our case is a string and applies the function to them, then calls the function with this result and the second argument and so on. This is a much more efficient way rather than manually recursing through the whole list. In the Haskell implementation I must place a type signature within the code that tells the function what the type of the variable is being passed to it. This is extra code which would not need to be added if Haskell could figure out the types on its own, but this way saves time as Haskell would not need to scan through the code to figure out what types correspond to what piece of code. In Prolog there is no need for a type signature for each predicate as Prolog lacks the understanding of different types. A big difference between the two implementations is the need for unknown variables in Prolog. In the Prolog implementation I must pass an unknown variable to the predicate, this will return the value for which the predicate is true, this could mean the

unknown variable could have multiple values but for our predicate this is not the case. This makes it more complicated to find the longest common prefix of a list of strings in Prolog, if you were unfamiliar with it.

Testing

I have tested the efficiency of both pieces of code by using the statistics keywords in Prolog and using the :set s function in Haskell, the results are as follows:

Haskell:

```
   lcp(["interview", "interrupt", "integrate", "intermediate"])
"inte"
(0.00 secs, 88,784 bytes)
=> (0.00 secs, 80,944 bytes)
```

Prolog:

```
11 ?- statistics(runtime, T1), lcpOfList(["interview", "interrupt",
"integrate", "intermediate"],L), statistics(runtime, T2).
T1 = [296, 296],
L = inte,
T2 = [296, 0] .

12 ?- statistics(memory, T1), lcpOfList(["interview", "interrupt",
"integrate", "intermediate"],L), statistics(memory, T2).
T1 = [18404, 0],
L = inte,
T2 = [20384, 0] .
```