Proposal for the Senior Project in Computer Science

Hunter Waldron

March 1, 2019

I propose that I create a new, experimental programming language for my senior project in computer science. This language will have a rather unique syntax that I have envisioned, inspired from Haskell and the Lisp family of programming languages. Although I intend to add as many useful features as I have time to, the foremost objective of this project will be having as elegant a syntax as possible. Because of this, I do not expect or even intend the finished programming language to be very practical, as any feature that cannot be implemented with sufficient elegance may simply not be implemented at all. However, I do expect that the finished project will be a reasonably usable programming language. That is, basic calculations, functions, recursion, and several other amenities will be present in the finished product. From this project I hope to gain more insight into how programming languages are parsed and evaluated. But most of all, I want to research how new and more esoteric programming language syntax and structure may lead to better program readability and perhaps even methodology.

In this programming language, there will be a type system including a built in integer, rational number, string, and Boolean type, user defined algebraic data types, as well as user defined unary operators, binary operators, and high order functions on all types. There is no user input and there are no mutable variables, meaning that this programming language will be purely functional. Before runtime, programs provided by the user will be checked for correctness. Correct programs must not reference any undefined or redefine any existing types, operators, or functions, and must not include expressions which contradict the form of any defined types. Furthermore, programs that import other source code files will have all imported files checked for correctness globally. If a program is correct, then it will be evaluated.

Because this programming language is purely functional, its evaluation is simple. At run time, functions may be partially applied and passed to other functions. This will be accomplished by passing the lexical scope that the function came from and also a list of pairs containing the variables that have been substituded and the value of that substitution. Existing data passed to functions does not need to be copied or changed due to its guaranteed immutability.

The program will run on the modern x86_64 Linux platform. Most of the project will be written in c. Some assembly will be required for System calls. No code written by anyone else will be used.

Below is a list of potential features with their worth to my grade. If I am able to earn 15 points, I deserve a C, for that will have yielded a working, usable product. If I am able to earn 20 points I deserve a B, as that will require substantial, complex work to achieve. If I can earn at least 25 points I deserve an A, as that will have produced a product that works about how I intended. There are 30 points possible.

Points	**Feature description**
8	Any correct user provided program can be correctly evaluated.
3	Program correctness is determined before run time.
3	Simple types: Functions, booleans, integers, rationals, and strings.
3	Advanced types: High order functions, lists of arbitrary types, and algebraic data types.
2	Programs may be imported into other programs.
2	Pattern matching for algebraic data types, lists, booleans, integers, and rationals.
2	User defined unary and binary operators.
2	Lexical scoping of variables.
2	Functions of several variables may be partially applied.
1	Any Function may be defined anonymously.
1	Arbitrary precision integers and rational numbers.
1	Programs can be entered interactively instead of only through files.