Functional Programming Language Implementation
TJHSST Computer Systems Lab 2009-2010

Jason Koenig
Latimer Prd. 5

October 28, 2009

1 Purpose and Scope

The purpose of my project is to develop a functional style programming language. This include both the language definition and a sample implementation. The language is similar to Lisp, but contains features to make it friendlier to imperative programmers. The initial version will be interpreted, but I expect to eventually at least partially compile code. The first interpreter has been written in Python, but the final implementation will be in C for speed. I also plan to include some optimization so that the language is not too slow. This can be everything from simple things such as constant propagation to complex things such as an optimizing G-code system.

One goal is to make the interpreter as small as possible, allowing the language to easily be embedded in other programs. This will allow my language to be used both on its own, and embedded as a scripting language like Python. Another goal is to create a language that allows both functional and imperative styles in the same language. Some of these features are similar to Lisp and Javascript, such as a definite execution order and allowance of local variables. I will also implement control structures such as while and foreach.

Beyond the implementation, I will also develop a series of tutorials and example programs that will assist in learning my new language. This will be important if my language is to become anything other than a toy language.

2 Background

There have been numerous functional languages over the years. The heaviest influences are from Lisp. Lisp has a definite execution order, and has support for resettable local variables. Lisp is very complex, however, and the interpreter is very large. It also has a complex and diversified family of languages, which makes it quite difficult to learn 'Lisp', rather than Scheme or Common Lisp or one of it's derivatives. Further, Lisp has a very large standard

library, which makes it difficult to port cleanly. My language would be focused on simplicity and speed, rather than on supporting every possible feature. This makes it easy to port.

Haskell shares more of a syntactic representation with my language. It however, is completely functional, which means no variable assignments. It is also lazy, which means computation is deferred until the last possible moment. Thus things like function side effects are not allowed. In languages like C++, sometimes expressions are evaluated simply for their side effect, like accessing a memory location to bring it into the cache. In Haskell, this is impossible, as simply accessing something is not enough to force its evaluation. This in turn forces the programmer into the functional style, which makes some operations, like input and output, harder. It also requires a shift in thought process to understand. I want to avoid this as much as possible in my language. By supporting impertive programming, I will ease more people into the functional style, and give my language a higher chance of success.

3 Procedure and Methodology

The first version of my interpreter will be written in the Python programming language because it is faster to prototype code in this language. The final version will be written in C or C++, because of the speed advantages. This will also allow me to create a C API, which is compatible with a large number of non-C languages.

The various aspects of my language will be tested as they develop. Once I reach a critical mass of code, in that the language is able to perform simple computations, the small tests that I use will be compiled into a corpus of programs and their appropriate output. By testing this corpus each time a new feature is implemented, I can ensure that previous parts of the program are still functional. Also part of this system would be a set of performance testing programs. When I introduce optimizations into the language, these will be helpful in analyzing the effect these optimizations have. The main corpus will also be helpful in making sure these optimizations do not change the behavior of code.

I will also test how easy my language is to learn, by teaching it to other students. By observing the results, I can evaluate both the design of the language and the effectiveness of the tutorials and documentation.

4 Expected Results

The programming language I will develop will be Turning complete. This means it wil be able to perform any computation. It will be a functional style language while being accessible to the average imperative programmer. I plan to develop bindings for some popular libraries, such as GTK and Berkley sockets.

I will produce both a language specification and an implementation of the interpreter. I also plan to develop tutorials on the use of my language, as well as several example programs of embedding the interpreter in other lanaguages. I will also develop a standard library, with support for things like file I/O, networking, and graphics.