Proposal for Senior Research: A Comparison of Lesser Known Programming Languages

David Colgan

May 4, 2011

1 Introduction

Most computer science majors and software developers have used Java and C. According to the TIOBE Index (`http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`), these two languages have consistently been among the most popular for general use. They are all established, well understood, and use the object oriented or imperative paradigm.

Most of the other top languages are fairly similar to Java and C. Languages like C#, PHP, Python, Perl, and Objective-C all use some combination of the procedural and object oriented paradigm. But are procedural and object oriented languages the best computer science has to offer for creating reliable, high performing software on a budget? Many lesser-known languages, a good number of which have heavy influence from the functional paradigm, claim increased programmer productivity, fewer errors, shorter programs, and greatly enhanced support for multicore processing.

2 Proposed Project

This project seeks to determine if Clojure, Forth, Erlang, Haskell, or J are compelling alternatives to the common procedural and object oriented languages most often used today in commercial environments.

These five languages are ones I deem interesting and have wanted to learn in the past. They are also all somewhat to very mind-bending, and several require a completely new approach to programming when compared to Java or C. The languages are:

- Clojure, a Lisp dialect on the Java Virtual Machine `clojure.org/`
- Forth, a stack-based language `www.forth.org/`
- Erlang, a concurrency-oriented language `www.erlang.org/`
- Haskell, a lazy, purely functional language `www.haskell.org/`

- J, an array language similar to APL `www.jsoftware.com/`

If I somehow get done with these languages and have more time, I could investigate OCaml, Scala, Lua, or Groovy.

3 Methods

I will take each of the five languages above and begin by spending some time learning how to program in it idiomatically. I will go through a well respected book and write small programs, as well as look at code written by respected programmers.

After I gain a good grasp on best practices in the language, I will implement a complex program using code that is as idiomatic as possible for the language.

As a control group, I will do the same thing in Java and C.

Using my observations from learning the languages and implementing the complex program, I will compare them based on:

- Ease of learning
- Ease of programming
- Lines of code
- Support for concurrency
- Features that assist large teams working together
- Performance
- Programmer enjoyment
- Development tools available (IDEs, debuggers, etc.)
- Libraries available

Lines of code is easy to compare. Support for concurrency will involve comparing the structures each language provides for such programming. Ease of learning could involve comparing the number of new concepts a procedural/OO programmer must learn to understand the new language. These criteria are tentative and will probably evolve as I move further into the research.

4 Comparison Program

The primary way I plan to compare the languages is by writing the same non-trivial program in each language. I want a system that is not too complicated, but also one that is not trivially simple. One day at family game night we played a dice game called Farkle (`http://en.wikipedia.org/wiki/Farkle/`). The game is simple but has a fair amount of decision making. Therefore for each language I would like to implement a system that both plays this dice game through a GUI interface, as well as evolve an optimal AI using a genetic algorithm system. Such a system would involve many different aspects that would explore each language's potential and features.

Farkle requires six dice. On each turn, the player rolls all six dice and removes combinations that are worth points. The following combinations are worth points:

- One 5 - 50 points
- One 1 - 100 points
- Three 1s - 300 points
- Three 2s - 200 points
- Three 3s - 300 points
- Three 4s - 400 points
- Three 5s - 500 points
- Three 6s - 600 points
- Four of a kind - 1000 points
- 1-6 Straight - 1500 points
- Three pairs - 1500 points
- Five of a kind - 2000 points
- Two triples - 2500 points
- Six of a kind - 3000 points

To score, the combination must be removed all on the same turn. As long as the player can remove at least one die that scores, they can then continue rolling. If the player ever cannot remove at least one die, they Farkle and lose all points for that turn. Therefore the strategy in the game comes from knowing when to stop rolling and which dice to set aside.

This game system will include a GUI frontend for human players and as a visualization of the GA. This will test the graphics capabilities of the language, as well as its library support. Since GAs are well suited for concurrency, it will be implemented to use multiple processing cores. The GA will therefore test the concurrency capabilities of the languages, as well as their symbolic manipulation power. The core game engine will allow human and AI players. If I am feeling especially adventurous, I may implement my own pseudo-random number generator, as a small, self-contained comparison of the languages.

5 Necessary Equipment

I will probably do most of this project on my own machine, since I will want to install many obscure programs. No other equipment will be necessary.

6 Proposed Timeline

It is important to note that some of these languages are bigger than others, and some are more difficult to learn and use than others. Therefore three weeks is a flexible amount of time. Some may take more, others may take less. For each language I will spend the first week or so reading books and reading others' source code. The second and third week will be devoted to building the Farkle system. The C and Java portions may require less time due to being more familiar to me.

6.1 Fall 2010

- Week 1 - Preparation
- Week 2-4 - C
- Week 4-7 - Java
- Week 8-10 - Clojure
- Week 11-13 - Forth
- Week 14 - Reporting and flexibility time

6.2 Spring 2011

- Week 1 - Preparation
- Week 2-4 - Erlang
- Week 4-7 - Haskell[?]
- Week 8-10 - J
- Week 11-14 - Reporting, presentation preparation, and flexibility time

I plan to complete a literature survey beginning now and maybe continuing through the C and Java phases of the project.