Evolving Compiled Programs

Eric C. Collom

ABSTRACT

This paper discusses using x86 assembly language and Java Bytecode to evolve programs. The benefits to evolving code at the Java bytecode and machine code level are: it's not limited by source code syntax, can evolve more than one type of language,it is possible to evolve the entire program.

Categories and Subject Descriptors

Software Engineering []: I will change category later

General Terms

I will change general terms later

Keywords

ACM proceedings, LaTeX, text tagging
I will change the keywords later

1. INTRODUCTION

I plan to focus on the topic of taking programs that aren't designed to be evolved and evolving them through Java bytecode and x86.

My two key papers will be "Flight of the FINCH through the Java Wilderness" [6] and "Automated Program Repair through the Evolution of Assembly Code" [7]. Sipper's et al. [6] research addresses if it is viable to evolve programs through Java bytecode. Forrest's et al. [7] research address if it possible to fix simple bug's of code in Assembly.

2. BACKGROUND

2.1 Evolutionary Computation

I will start off my background by describing Evolutionary computation. I will explain terminology such as mutation, crossover, fitness value, and tournament (and other things). I will also explain how they work individually and together as a big picture [1].Also, Forrest et al. [7] have a well done background on EC so I might use it as well. Sipper and Orlov [6] have a good description on crossover.

UMM CSci Senior Seminar Conference Morris, MN.

2.2 Assembly

I will explain enough about assembly and how it works so that the reader can understand the rest of the paper. For background on this so far I only have [3] as a reference which I need to actually read and understand yet. I feel like I will need more but I'm still not sure.

2.3 Java Bytecode

Java source code is first compiled to platform independent Java bytecode. The Java virtual machine (JVM) then checks the bytecode for errors. If it finds an error it throws an exception if not it begins to interpret the bytecode until it decides it should compile it into assembly [6].

3. WHY EVOLVE JAVE BYTECODE AND X86

It is beneficial to evolve Java bytecode and x86 assembly because the entire program can be evolved and not just small parts. FINCH, for example, can evolve a program that is nowhere near the final solution and is lacking much of the framework to solve the problem [6].

Evolving the entire program at this level is much simple that evolving at the source code level.This is because the bytecode and x86 grammar sets are a lot smaller and with source code there are lots of syntax constraints [6] [7].

If we would want to try to solve this problem by evolving at the source code level then basically a new compiler would need to be developed in order for the evolving program to interpret the syntax and then act accordingly [6].

Any program that compiles into Java bytecode or x86 can be evolved. This provides a large pool of languages [6] [7]. For example, Scala, Groovy, Jython, Kawa, JavaFx Script, and Clojure all compile to Java bytecode [6]. Thus, FINCH could actually evolve programs written in these languages.

4. HOW THE FINCH EVOLVES CODE

The main EC tool that FINCH used was crossover which it applies 90% of the time. It applied crossover to two parents programs that already compile. It then checks if the offspring compiles. If it doesn't it then tries to produces another good offspring. It does this until either a good offspring is produced or a specified amount of attempts have been made [6].

After the offspring population is generated then tournament selection is performed to select the next generation of programs [6].

Deals with halting issues by counting function executions and terminating after the count reaches a predetermined amount.

5. USING ASSEMBLY FOR SOFTWARE REPAIR

Forrest et al. [7] used tests to evaluate the fitness. The initial individual usually had five tests that it passed and one test that it didn't. Large to small sized programs were used for evolution. They evolved programs written in C, java, and Haskell.The program would stop evolving the program once an individual was found that passed all six tests. A 90% chance Mutation was used and crossover the rest of the time [7].

6. RESULTS

6.1 Assembly Results

The results for repairing programs through assembly language looked promising. It was shown that evolving code at the assembly to fix small bugs such as infinite loops or buffer overflows [7].

It was discovered that complex operators were very ineffective and would slow down the process [7].

6.2 FINCH Results

The results using the FINCH [6] to evolve Java bytecode also looked promising. Their results showed that FINCH was just as fast and dependable as traditional EC methods. They were able to show that evolving programs at bytcode level to solve traditional EC problems is feasible and works.

6.3 Future Work

Would be nice to see the FINCH solve problems with a much larger source code. They could also solve issues where recursive methods were evolved that didn't terminate but were returned as the methods with the highest fitness. This was due to the way they set up FINCH to deal with the halting issue.

For the assembly project the results were not consistent from language to language even though they were all evolved at assembly language level. Also, would be nice to see them work on programs that are more object oriented heavy.

7. CONCLUSION

In plan to use the following sources: [7] and [6] as my main sources. I will possibly use [4] and [5] as more information on evolving programs for different EC purposes such as problem solving or debugging. I will use [3] and [2] as background into x86 assembly language and Java bytecode. I also have [1]for background on EC.

8. REFERENCES

[1] *A Field Guide to Genetic Programming.* 2008.
This will prived general definitions/descriptions for EC terminology. .

[2] *The Java Language Specification.* Oracles America, Inc. And/or affiliates, 500 Oracle Parkway, Redwood City, California 94065, U.S.A., 2013.
This provides some needed background on the JVM, compiler, and Java bytecode. It is too long to read it all but will be a good reference for any questions about the JVM .

[3] x86 assembly language. January 2014.
This provides background for a general understanding of x86 assembly language .

[4] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark.

[5] W. B. Langdon and M. Harman. Genetically improving 50000 lines of c+. *RN*, 12(09):09, 2012.
This is research notes about using using EC to evolve and improve C++ programs. Since it is research notes it is a bit lengthy. It also mentions evolving things through bytecode. Not sure if I will use yet .

[6] M. Orlov and M. Sipper. Flight of the finch through the java wilderness. *Evolutionary Computation, IEEE Transactions on*, 15(2):166–182, April 2011.
This will be one of the backbones of my paper. Talks about a program called FINCH that the authors developed. Finch is used to evolve programs at the Java bytecode level. This paper Has lots of information and they have about six examples where they evolved programs using FINCH .

[7] E. Schulte, S. Forrest, and W. Weimer. Automated program repair through the evolution of assembly code. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 313–316, New York, NY, USA, 2010. ACM.
This will be a main source for my paper. It talks about evolving programs at assembly language level and its benefits. These authors focus more on doing this to use EC to debug, refactor, and repair programs. It also breifly mentions FINCH .