# FINDING COMPILER BUGS VIA CODE MUTATION: A CASE STUDY

Chiran Sachintha Abeygunawardana

219176C

Degree of Master of Science

Department of Software Engineering

Faculty of Engineering

University of Moratuwa

Sri Lanka

July 2023

# FINDING COMPILER BUGS VIA CODE MUTATION: A CASE STUDY

Chiran Sachintha Abeygunawardana

219176C

Thesis submitted in partial fulfillment of the requirements for the degree

Master of Science in Software Engineering

*Department of Software Engineering*

Faculty of Engineering

University of Moratuwa

Sri Lanka

July 2023

# DECLARATION

I declare that this is my own work, and this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Chiran Sachintha                                                    Date: 30/07/2023

The above candidate has carried out research for the master's thesis under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Professor Indika Perera

Signature of the Supervisor:                                                    Date: 30/07/2023

## DEDICATION

I dedicate my thesis to my parents in appreciation of their endless love, support, and encouragement throughout my pursuit of higher education. I hope this achievement will fulfil the dream they imagine for me.

## ACKNOWLEDGEMENT

I want to express my sincere gratitude to my supervisor, Prof. Indika Perera, for his efforts, ongoing guidance, and support. I feel incredibly fortunate to have him as my supervisor because his advice improved my study. His encouragement led me to successfully complete this research.

I also want to thank all my friends for supporting me during this study process with their help and encouragement.

And most significantly, without my family's love and support, none of this would have been possible. I am appreciative of my family for supporting me during this time and encouraging me to finish the research.

# ABSTRACT

Compiler errors can cause a variety of problems for software systems, including unexpected program behavior, security flaws, and system failures. These defects can be brought on by a number of things, including improper data type handling, poor code creation, and wrong code optimization. Compiler defects can be difficult to spot due to their complexity and, if ignored, can have severe effects. So, Identifying compiler defects is a crucial and difficult undertaking because it is difficult to produce reliable test programs.

One of the most used software testing techniques for finding bugs and vulnerabilities is fuzzing. Fuzzing is the process of generating numerous inputs to a target application while keeping an eye out for any anomalies. Among fuzzing techniques, the most recent and promising methods for compiler validation are test program generation and mutation. Both methods have proven to be effective in identifying numerous problems in a variety of compilers, although they are still constrained by the techniques' use of valid code creation and mutation methodologies.

Code mutation is a method that has grown in favor recently since it can find bugs that standard testing can forget. This technique involves performing minor adjustments to a program's source code to make versions of the original code, which are then compiled and evaluated to see if they deliver the desired outcomes. It is a sign that there might be a compiler issue if the output of the altered code differs from the output of the original code.

Current mutation-based fuzzers randomly alter a program's input without comprehending its underlying grammar or semantics. In this study, we proposed a novel mutation technique that mutates the existing program while automatically understanding the syntax and semantic rules. Any type of compiler can be verified using the suggested method without regard to the semantics of the language. With that we can use this approach to test various other compilers without depend on the syntax of that language. We focus on evaluating the Ballerina compiler to the language syntax and semantics because Ballerina is a relatively new programming language.

In this work initially we construct a test suite from the present testcases of that language and developed a syntax tree generator which can identify the syntax of that language and then developed semantic generator which can identify semantic of that language. With that we are able to mutate the existing test cases using our generator. Furthermore, we have analyzed the performance of our model with the number of test cases which use to train our model and the number of tokens in the generated file.

**Keywords**: Compiler testing, random testing, random program generation, Automated testing

# TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

EMI – Equivalent Module Input

JVM – Java Virtual Machine

et al – And Others

GCC – GNU Compiler Collection

LLVM – Low Level Virtual Machine

OpenJDK – Open Java Development Kit

# LIST OF APPENDICES