

Static Symbolic Execution in SVF

by Guanqin ZHANG

Thesis submitted in fulfilment of the requirements for
the degree of

Master of Analytics (Research) in Computer Science

under the supervision of Yulei SUI

University of Technology Sydney
Faculty of Engineering and Information Technology

Jan 2022

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, *Guanqin ZHANG* declare that this thesis, is submitted in fulfilment of the requirements for the award of *Master of Analytics (Research) in Computer Science* in the *Faculty of Engineering and Information* at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Production Note:

Signature: Signature removed prior to publication.

Date: Jan 25th, 2022

Abstract

This thesis presents the implementation of a static symbolic execution (**SSE**) tool on top of SVF framework. SSE is a well-studied technique for program testing and verification, which is based on collecting the program execution paths and exploring the program behaviours in the form of the constraints. SSE relies on SVF-IR language to translate the program into logical formulas of a particular rules-based form which supplies useful program properties for use in the program analysis. SSE maintains all elements of program running data and its manipulations. More precisely, SSE traverses on program control flow graph and modulates program behaviours, e.g., buffer-over-flow is an anomaly in program, which can be identified that the data overruns the buffer's boundary and overwrites adjacent memory locations. Experiments conducting our implementation of SSE indicates many benefits by using symbolic execution, e.g., depth-first searching on particular paths to mimic real program execution process, or finding certain program anomalies based on translating rules.

Table of Contents

List of Figures	v
List of Tables	v
1 Introduction	1
1.1 Motivation	1
1.2 Example	2
1.3 Research Contribution	4
1.4 Framework Overview	4
1.5 Thesis Structure	6
2 Preliminaries	7
2.1 Symbolic execution	7
2.2 Control & Data dependence analysis	8
2.3 LLVM IR	8
2.4 SVF	10
2.4.1 Domain	10
2.4.2 LLVM-like language	11
3 Path Collection	14
3.1 Inter-procedural control flow graph (ICFG)	14

3.2	ICFG Path	16
4	Symbolic execution	20
4.1	Introduction	20
4.2	Existing symbolic execution tools	21
4.3	An Overview of SSE	22
4.4	Memory Management	23
4.5	Path Solving	26
4.6	Satisfiability modulo theories	29
5	Validation	31
5.1	Testing	31
5.2	Optimization and Future Work	37
5.3	Conclusions	38
	Bibliography	40

List of Figures

FIGURE	Page
1.1 C code example and its control flow paths	3
1.2 The overview of the proposed SSE -tool in thesis.	5
3.1 Context Sensitive Case Example	17
5.1 Example (1) Integer computes	32
5.2 Example (2) One-level pointers	33
5.3 Example (3) Multiple-level pointers	33
5.4 Example (4) Structure	35
5.5 Example (5) Call	37

List of Tables

TABLE	Page
2.1 Program Variables & Domain	10
2.2 LLVM-like language	11

3.1	ICFG nodes and edges	15
4.1	Symbolic execution <i>Expr</i> syntax	24
4.2	Translation Rules	24
5.1	Example (1) SSE Translation	32
5.2	Example (2) SSE States Transitions	34
5.3	Example (3) SSE States Transitions	35
5.4	Example (4) SSE States Transitions	36
5.5	Example (5) SSE States Transitions	38