Symbolic Execution(Working title)

Aarhus Universitet

Søren Baadsgaard

February 3, 2019

Abstract

Contents

Chapter 1

Introduction

Chapter 2

Summary of theory

Chapter 3

Basic symbolic execution for the *SImPL* language

3.1 description

In this chapter we will describe the process of implementing symbolic execution for a simple imperative language called *SImPL* .

3.2 Introducing the *SImPL* language

SImPL **(Simple Imperative Programming Language)** is a small imperative programming language, designed to highlight the interesting use cases of symbolic execution. The language supports only one type, namely the set integers $\mathbb{N}$. Furthermore we will interpret 0 as *false* and any other integer as *true*. *SImPL* supports basic variables that can be assigned the value of any expression, as well as basic branching functionality through an **If - Then - Else** statement. Furthermore it allows for looping through a **While - Do** statement.

We will describe the language formally, by the following Context Free Grammar:

$\langle int \rangle$ ::= 0 | 1 | -1 | 2 | -2 | ...

$\langle var \rangle$ ::= a | b | c | ...

$\langle exp \rangle$ ::= $\langle int \rangle$
| $\langle var \rangle$
| $\langle exp \rangle + \langle exp \rangle \mid \langle exp \rangle - \langle exp \rangle \mid \langle exp \rangle * \langle exp \rangle \mid \langle exp \rangle \ / \ \langle exp \rangle$
| $\langle exp \rangle > \langle exp \rangle \mid \langle exp \rangle == \langle exp \rangle$
| ($\langle exp \rangle$)

$\langle stm \rangle$ ::= $\langle exp \rangle$
| $\langle var \rangle = \langle exp \rangle$
| $\langle stm \rangle \ \langle stm \rangle$
| if $\langle exp \rangle$ then $\langle stm \rangle$ else $\langle stm \rangle$
| while $\langle exp \rangle$ do $\langle stm \rangle$
|

where $+, *, -, /$ denotes the usual arithmatic operators on integers, and $>$, $==$ denotes the comparison-operators of *greater-than* and *equal-to* respectively. When interpreting a comparison-operator we will return 0 for *false* and 1 for *true*. Note that we have defined the language such that a program is simply one or more statements, and that every statement will return some value. In the case of an assign-statement, we simply return the value of the expression on the right hand side.

Chapter 4

Further extensions

Chapter 5

Conclusion

Appendix A

Source code

Appendix B

Figures