A Survey of
Symbolic Executions Techniques

Hallet Adrien Sens Loan

October 9, 2018

Abstract

1 Introduction

1.1 (Attempting) A definition

The first occurences of symbolic execution described the then-new method as a middle ground [3] between the two most-used method of its time. On one hand, program testing (*e.g.: unit testing*) can not always detect a fault in a program and producing a correct test sample and proving that it indeed is correct is not that easy. On the other hand, program proving can indeed ensure that a program is correct from its entry point to the result but it heavily relys on the proof definitions by the programmer and the formal definition of the problem.
Nowadays, symbolic execution is both described as (part of) the core of many modern techniques to software testing [4] and an effective way to create tests suites with extensive coverage. [1]

1.2 The concept

2 History

3 Methods

3.1 Concolic execution

The name ”concolic” is a portmanteau of the words ”concrete” and ”symbolic”, the idea of this testing method is to mix symbolic execution alongside concrete ones.

This technique concept was first introduced on 2005 [2]. Since then the idea was further extended and combined with other testing techniques.

4 Tools and languages

4.1 *DART* : Directed Automated Random Testing

DART is presented as a tool for automatically testing software using concolic testing method (see section 3.1). It was introduced in 2005 making it the first the first tool to be created using concolic techniques.

4.1.1 Methodology

DART combines three main techniques [2] in order to automate unif testing for a particular software :

1. An automated extraction of the interface of a program with its external environment using static source-code parsing
2. An automatic generation of a test driver for this interface that performs random testing to simulate the most general environment the program can operate in
3. A dynamic analysis of how program behaves under random testing and automatic generation of new test inputs to direct systemically the execution along alternatives program paths

4.1.2 Key strength/originality

The main strength of DART is that testing can be performed completely automatically on any program that compiles – there is no need to write any test driver or harness code.
During testing, DART detects standard errors such as program crashes, assertion violations, and non-termination.
DART provides an attractive alternative approach to static analyzers, because it is based on high-precision dynamic analysis instead, while being fully automated as static analysis. The main advantage of DART over static analysis is that every execution leading to an error that is found by DART is guaranteed to be sound. Two areas where we expect DART to compete

especially well against static analyzers are the detection of interprocedural bugs and of bugs that arise through the use of library functions.

DART is overall complementary to static analysis since it has its own limitations, namely the computational expense of running tests and the sometimes limited effectiveness of dynamic test generation to improve over random testing.

5 Conclusions

References

[1] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: Three decades later. 56:82–90, 02 2013.

[2] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. In *ACM Sigplan Notices*, volume 40, pages 213–223. ACM, 2005.

[3] J.C. King. A new approach to program testing. 10:228–233, 06 1975.

[4] David Trabish, Andrea Mattavelli, Noam Rinetzky, and Cristian Cadar. Chopped symbolic execution. In *International Conference on Software Engineering (ICSE 2018)*, 5 2018.