JOHANNES KEPLER
UNIVERSITY LINZ

DI David Leopoldseder
Institute for System Software

P +43 732 2468 4356
F +43 732 2468 4345
david.leopoldseder@oracle.com

Office:
Karin Gusenbauer
Ext. 4342
karin.gusenbauer@jku.at

Master's Thesis

Symbolic Execution on GraalVM

Student: Thomas Pointhuber, Bsc.

Advisor: DI David Leopoldseder

Start Date: Oct. 7, 2019

Symbolic Execution[1] is a technique that abstracts inputs of a program into symbolic values, to allow the analysis of its behavior. For example, this allows tools to automatically find bugs like buffer overflows or generate inputs to simulated the execution of a specific execution path. There are several symbolic execution tools available for Java, e.g. Symbolic PathFinder[2], JBSE[3], janala2[4] and angr[5]. A well-known implementation outside the Java ecosystem is KLEE[6], which is based on the LLVM[7] ecosystem.

GraalVM[8] is a runtime environment written in Java that includes state-of-the-art just-in-time compilation as well as APIs for polyglot language development. GraalVM contains the Truffle[9] framework, a language implementation framework which supports a straight-forward integration of new programming languages. This can be done by implementing a domain-specific abstract syntax tree, which is then executed on GraalVM. This leads to competitive execution performance as well as the ability to use integrated debugging and cross-language interoperability support.

While there exist a number of symbolic execution frameworks for Java, those are not usable for Polyglot languages as supported by GraalVM. Most of the symbolic execution tools for Java, are based on Java bytecode. Truffle, however, represents language semantics with ASTs allowing symbolic execution across language boundaries. With the integration of a symbolic execution framework into GraalVM itself, it would make it possible to do efficient symbolic execution of Java as well as of cross-language environments.

The goal of this thesis is to implement a symbolic execution framework on top of GraalVM for Java programs represented via Graal compiler IR. The implementation should support the Java spec and allow the symbolic execution of medium-sized programs like the ones found on "The Computer Language Benchmarks Game"[10].

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at
DVR 0093696

The scope of this thesis is as follows:

- allow the symbolic execution of medium complex java programs
- support major parts of the Java API
- automated testing of API functions for exception states
- input generation to reach a specified target state
- test case generation which focuses on increased path coverage

Out of scope of the project which will be refined over time is:

- support of non-Java languages on GraalVM (Truffle)
- Java API parts which are hard to map into symbolic execution. E.g. reflection, JNI, network, crypto,...
- multithreading
- loop invariant
- java modeling language support

[1] King, J. C. (1976). Symbolic execution and program testing. *Communications of the ACM*, *19*(7), 385-394.

[2] Păsăreanu, C. S., & Rungta, N. (2010, September). Symbolic PathFinder: symbolic execution of Java bytecode. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (pp. 179-180). ACM.

[3] Braione, P., Denaro, G., & Pezzè, M. (2016, November). JBSE: a symbolic executor for Java programs with complex heap inputs. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 1018-1022). ACM.

[4] janala2: a concolic testing engine for Java. Retrieved from https://github.com/ksen007/janala2

[5] Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., ... & Vigna, G. (2016, May). Sok:(state of) the art of war: Offensive techniques in binary analysis. In *2016 IEEE Symposium on Security and Privacy (SP)* (pp. 138-157). IEEE.

[6] Cadar, C., Dunbar, D., & Engler, D. R. (2008, December). KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *OSDI* (Vol. 8, pp. 209-224).

[7] Lattner, C., & Adve, V. (2004, March). LLVM: A compilation framework for lifelong program analysis & transformation. In Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization (p. 75). IEEE Computer Society.

[8] Würthinger, T., Wimmer, C., Wöß, A., Stadler, L., Duboscq, G., Humer, C., ... & Wolczko, M. (2013, October). One VM to rule them all. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software* (pp. 187-204). ACM.

[9] Wimmer, C., & Würthinger, T. (2012, October). Truffle: a self-optimizing runtime system. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity* (pp. 13-14). ACM.

[10] The computer language benchmarks game. https://benchmarksgame-team.pages.debian.net/benchmarksgame