JOHANNES KEPLER
UNIVERSITY LINZ

Master's Thesis

Symbolic Execution for Compiler Optimizations on the GraalVM

Student:	Sebastian Kloibhofer (01555702)
Advisor:	Prof. Mössenböck, Dr. Leopoldseder
Start date:	01.10.2019

Linz, October 1, 2019

o.Univ.-Prof. Dr. Dr.h.c.
Hanspeter Mössenböck
Institute for System Software

P +43 732 2468 4340
F +43 732 2468 4345
hanspeter.moessenboeck@jku.at

Secretary:
Birgit Kranzl
Ext. 4341
birgit.kranzl@jku.at

GraalVM[1] is a Java virtual machine that supports the execution of dynamic languages such as JavaScript, Python, Ruby etc. as well as static languages like LLVM bitcode. It allows the programmer to express guest-language semantics in Java AST implementations via the Truffle framework. The Graal compiler—GraalVM's core component—is a dynamic JIT compiler that generates highly optimized code using speculative optimizations and the concepts of deoptimization [1].

To do so, the GraalVM compiler transforms a program's bytecode to an intermediate representation—the Graal IR—organized as a graph. This representation simplifies subsequent optimizations such as Constant Folding, Partial Escape Analysis and Method Inlining. Still, those optimizations are mostly limited to the information gained from profiling as well as static analysis methods [2, 3].

In order to further optimize the IR, the concept of symbolic execution should be applied to guide compiler optimizations. This thesis should combine symbolic execution with compiler optimizations in GraalVM by building a Graal IR model that is mapped to logic formulas for constraint solvers. This model can then be used by the compiler to explore different program paths and check for optimization potentials such as algebraic reductions, redundant operations or infeasible branches and modify the graph accordingly.

Symbolic execution is a prominent technique in software verification and analysis to identify program flaws and generate test cases. The general idea of symbolic execution is to simulate the execution of a program and to form path conditions that describe the possible paths through the control flow [4]. Inputs are treated as so-called "symbolic" values which represent unknown variables that are used for building formulas to model the semantics of the program. Modifications and constraints on symbolic variables are stored in a symbolic memory. Typically, these structures are then passed on to a SAT / SMT solver to determine whether a path (under the given conditions and with the given memory contents) is feasible and—if so—to generate a satisfiable assignment (i.e. input values for a method that lead to the target result) [5, 6, 7].

While logic systems are already used to formally verify certain compiler techniques [8, 9], the application of symbolic execution in the context of compiler optimizations is rare: There are publications on partial evaluators

[1] https://www.graalvm.org/

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

for JavaScript—namely Prepack[2]—and other approaches that propose the use of symbolic execution to improve C preprocessors [10, 11, 12]. However, to our knowledge there is no approach utilizing symbolic execution to guide optimizations in dynamic compilers.

The goals of this thesis are as follows:

1. Implementation of a framework in order to support compiler optimizations via symbolic execution of the low-tier Graal IR
2. Implementation of algebraic simplifications on the graph using symbolic execution
3. Evaluation of the applied optimizations with respect to compilation overhead and runtime benefits on the SpecJVM2008[3], Scalabench[4], DaCapo[5] and Renaissance[6] benchmarks

Non-goals are:

1. Implementation of symbolic simplifications for common low-tier Graal IR nodes
2. Implementation of a (low-tier) memory model to enable reasoning about memory operations which is required for memory based optimizations
3. Development of an API and test suite to simplify the creation of custom optimizations

References

[1] Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko. One VM to Rule Them All. In Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Onward! 2013, pp.187-204, New York, NY, USA, 2013. ACM. event-place: Indianapolis, Indiana, USA.

[2] Gilles Duboscq, Thomas Würthinger, Lukas Stadler, Christian Wimmer, Doug Simon, and Hanspeter Mössenböck. An Intermediate Representation for Speculative Optimizations in a Dynamic Compiler. In Proceedings of the 7th ACM Workshop on Virtual Machines and Intermediate Languages, VMIL '13, pp.1-10, New York, NY, USA, 2013. ACM. event-place: Indianapolis, Indiana, USA.

[3] Gilles Marie Duboscq. Combining speculative optimizations with flexible scheduling of side-effects. PhD thesis, Linz, April 2016, 2016.

[4] James C. King. Symbolic Execution and Program Testing. CACM, 19(7):385-394, July 1976.

[5] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, pp.337-340. Springer Berlin Heidelberg, 2008.

[6] Robert Brummayer and Armin Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In Stefan Kowalewski and Anna Philippou, editors, Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, pp.174-177. Springer Berlin Heidelberg, 2009.

[7] C. Cadar, P. Godefroid, S. Khurshid, C. S. Pasareanu, K. Sen, N. Tillmann, and W. Visser. Symbolic execution for software testing in practice: preliminary assessment. In 2011 33rd International Conference on Software Engineering (ICSE), pp.1066-1071, May 2011.

[2] https://prepack.io/
[3] https://www.spec.org/jvm2008/
[4] http://www.scalabench.org/
[5] http://dacapobench.org/
[6] https://renaissance.dev/

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

[8] David Lacey, Neil D. Jones, Eric Van Wyk, and Carl Christian Frederiksen. Compiler Optimization Correctness by Temporal Logic. Higher-Order and Symbolic Computation, 17(3):173-206, September 2004.

[9] Florian Merz, Stephan Falke, and Carsten Sinz. LLBMC: Bounded Model Checking of C and C++ Programs Using a Compiler IR. In Rajeev Joshi, Peter Müller, and Andreas Podelski, editors, Verified Software: Theories, Tools, Experiments, Lecture Notes in Computer Science, pp.146-161. Springer Berlin Heidelberg, 2012.

[10] Sumeyye Suslu. JSSpe: A Symbolic Partial Evaluator for JavaScript. Thesis, The University of Texas at Arlington, April 2018.

[11] Sümeyye Süslü and Christoph Csallner. SPEjs: A Symbolic Partial Evaluator for JavaScript. In Proceedings of the 1st International Workshop on Advances in Mobile App Analysis, A-Mobile 2018, pp.7-12, New York, NY, USA, 2018. ACM. event-place: Montpellier, France.

[12] Ying Hu, Ettore Merlo, Michel Dagenais, and Bruno Lagüe. C/C++ Conditional Compilation Analysis Using Symbolic Execution. In Proceedings of the International Conference on Software Maintenance (ICSM'00), ICSM '00, pp.196-, Washington, DC, USA, 2000. IEEE Computer Society.

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