VeriTest: Automatic Verification of Compiler Optimizations

Achmad Afriza Wibawa
School of Electrical Engineering and Computer Science
The University of Queensland, Qld., 4072, Australia

ABSTRACT

VeriTest aims to provide feedback to GraalVM developers on proposed optimizations encoded in Veriopt's domain specific language (DSL), integrating into their development workflow without requiring expertise in formal proofs. By leveraging Isabelle as the proof engine, we can provide a comprehensive analysis on the optimization rule, providing fast feedback – either counterexamples or proofs – to the developer. The implementation utilizes the Isabelle Client - Server interface to ensure concurrent and stateless processing. During evaluation of existing optimization rules, VeriTest demonstrated its capability in finding several malformed or erroneous optimization rules, while also utilizing existing lemmas to find proofs.

I. INTRODUCTION

This paper introduces VeriTest: an Automated Testing Framework for GraalVM's expression optimizations. VeriTest builds upon the work done on the formal specification of GraalVM's Intermediate Representation (IR) [1] and defining optimization rules as a term rewriting rule using a Domain Specific Language (DSL) [2].

The goal of VeriTest is to provide feedback to GraalVM [3] developers on their proposed optimizations and integrate it into their development workflow. This would make it easy for developers to use the tool *as you go*, without being a *"proof expert"*. To achieve that, we leverage automation tools built in interactive theorem provers such as Isabelle [4]. This is inspired by the work done on LLVM by Alive [5], where peephole optimizations would be verified to be semantically correct by leveraging SMT solvers [5, Sec. 3.1.1].

II. ISABELLE AS PROOF ENGINE

Isabelle is an interactive theorem prover that utilizes syntax translations of a theory definition into a set of inference rules that can be automatically reasoned within the system [4]. The reasoning is done by functions that work on a proof state called tactics. Tactics either output a direct proof towards the goal or break them down into smaller sub-goals in a divide-and-conquer manner. VeriTest focuses on several proof tools inside Isabelle: Sledgehammer [4, Sec. 3], Quickcheck [4, Sec. 4], and Nitpick [4, Sec. 5].

III. VERIOPT'S DSL

Veriopt's DSL enables GraalVM developers to express optimizations as rewriting rules in a Java syntax similar to GraalVM's IR within Isabelle [2]. Webb et al. [2] describe that there are 2 proof obligations that must be met for the side-effect-free optimization to be correct: 1) proof that the optimization rule will terminate; 2) proof that each pass of the optimization rule will result in a refinement of the expression.

IV. METHODOLOGY

In order to provide feedback for GraalVM developers, VeriTest would need to verify whether the optimization rule: 1) is obviously false – a counterexample exists; 2) is obviously true – a proof can be found; 3) require manual proving by *"proof experts"*.

To leverage Isabelle, VeriTest utilizes Isabelle Client - Server [6] interactions as a method of interfacing with Isabelle tools. Isabelle Server acts as the core Isabelle process that allows theorems and all the required facts to be loaded up and processed in parallel by Isabelle/ML which is ideal for VeriTest [6, Ch. 4.2.6]. As theorems can interfere with one another, VeriTest ensures that each theorem is mutually exclusive by separating the sessions that they are run on, making them essentially stateless.

Figure 1 demonstrates the architecture of VeriTest. As there are multiple ways of interacting with Isabelle Server [6, Ch. 4.2], VeriTest abstracts the interaction by utilizing a bridge design pattern. The abstraction is implemented by Isabelle Process, which is a proxy for an Isabelle Client subprocess.

Utilizing the proof tools provided by Isabelle yields a comprehensive analysis of the optimization rule, as we're making use of a battle-tested automated theorem prover. Passing optimization rules into Isabelle Client is done in parallel, utilizing asynchronous functions managed by a thread pool inside the system. In order to provide fast feedback, these asynchronous functions implements a circuit-breaker pattern when either a proof or a counterexample is found.

A. Finding Counterexample

In order to find counterexamples, we utilize Nitpick and Quickcheck from Isabelle. Optimization rules are represented as an Isabelle/HOL data type, the expression is fully translated into conjunctions which can be inferred as satisfiable or not [2, Sec. 2].

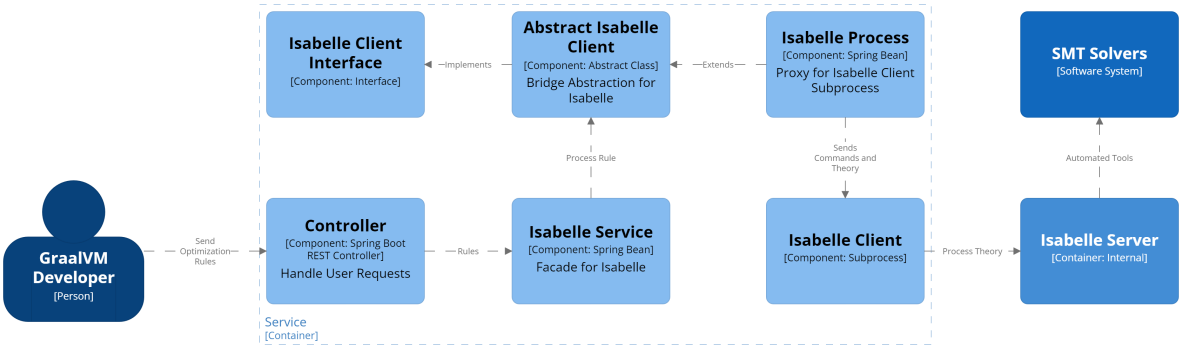

Figure 1. Solution Architecture of VeriTest

B. Finding Proof

Sledgehammer is invoked recursively in an attempt to verify the optimization rule. As after Sledgehammer is invoked, there may be multiple possible proofs available, and it may not completely prove all proof obligations. Thus, we use the partially proven optimization rule and re-invoke Sledgehammer, until either it succeeds or fails to prove all proof obligations.

C. Limitations

Unexpectedly, there is a key flaw inside Isabelle Server. Kobschätzki [7] notes that Isabelle Server triggers a race condition with multiple users and sessions, unless a delay of several seconds is placed in between subsequent commands. Isabelle Server's source code also confirms that each command invocation doesn't support concurrent use. As such, this introduces a bottleneck for parallel processing of optimization rules.

V. Evaluation

Result	# Rules	Mean $\pm$ *SD*
Failed	59	87.06 $\pm$ 49.42
Found Auto Proof	7	37.73 $\pm$ 3.92
Found Proof	32	82.91 $\pm$ 29.43
Found Counterexample	1	40.79 $\pm$ 4.02
Malformed	13	37.96 $\pm$ 4.02
No Subgoal	2	37.93 $\pm$ 3.38

Figure 2. Evaluation of each existing Veriopt optimization rules based on runtime (in seconds)

Evaluation is done by exporting the existing Veriopt's expression canonicalization optimization rules and iterating them five times as a benchmark on an AMD Ryzen 9 7900X processor with 64GB of RAM inside Windows 10 WSL environment. Figure 2 describes the result of the evaluation.

Our findings suggests that there is a baseline runtime for every verification of an optimization rule, since each subsequent Isabelle command invocation is stateless. Additionally, the limitation of Isabelle Server introduces a bottleneck to the system. This is demonstrated by the consistent runtime of finding automatic proofs, counterexamples, and malformed rules.

The evaluated runtime suggests a significant variation of runtime for some results. This is due to VeriTest attempting to exhaust every possible proof and counterexample for the optimization rule. Failure to find a proof suggests that it *may* be able to be proven with adequate expertise in formal proofs. As the complexity of optimization rules vary, the depth of the recursive tree depends on the proof obligations that Sledgehammer can prove.

Furthermore, we found that VeriTest is capable of utilizing existing lemmas inside Veriopt's current theory base to find proofs for the optimization rule. This implies that, as the collection of theories grow, the percentage of optimization rules that can be automatically proven is improved.

Surprisingly, some of the existing, commented out, optimization rules are malformed. After investigation, we found that it is malformed due to type unification errors over the IR terms.

References

[1] B. J. Webb, M. Utting, and I. J. Hayes, "A formal semantics of the GraalVM intermediate representation," in *Automated Technology for Verification and Analysis*, ser. LNCS, Z. Hou and V. Ganesh, Eds., vol. 12971, Oct. 2021, pp. 111–126.

[2] B. J. Webb, I. J. Hayes, and M. Utting, "Verifying term graph optimizations using Isabelle/HOL," in *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2023, p. 320333.

[3] Oracle, "GraalVM: Run programs faster anywhere," 2020. [Online]. Available: https://github.com/oracle/graal

[4] J. C. Blanchette, L. Bulwahn, and T. Nipkow, "Automatic Proof and Disproof in Isabelle/HOL," in *Frontiers of Combining Systems*, ser. LNCS, C. Tinelli and V. Sofronie-Stokkermans, Eds., 2011, vol. 6989, pp. 12–27.

[5] J. Lee, C.-K. Hur, and N. P. Lopes, "AliveInLean: A verified LLVM peephole optimization verifier," in *Computer Aided Verification*, I. Dillig and S. Tasiran, Eds., 2019, pp. 445–455.

[6] M. Wenzel, "The Isabelle System Manual." [Online]. Available: https://isabelle.in.tum.de/dist/Isabelle2023/doc/system.pdf

[7] J. Kobschätzki, "Unexpected Behavior with Isabelle Server 2023," Jan. 2024. [Online]. Available: https://lists.cam.ac.uk/sympa/arc/cl-isabelle-users/2024-01/msg00006.html