VeriTest: Automated Testing Framework for Veriopt's DSL

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 their optimizations written in Veriopt's domain specific language (DSL). Integrating seamlessly into their development workflow, without requiring expertise in formal proofs. By leveraging Isabelle as the proof engine, we can provide a comprehensive analyisis on the optimization rule, providing fast feedback – either counterexamples or proofs – to the developer. The implementation utilizes Isabelle Client - Server interactions as an abstraction that ensures 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 complement finding 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 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"*. In order to achieve that, we can leverage the 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 could 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 allows GraalVM developers to express optimizations as a rewriting rule, using a Java syntax similar to GraalVM's IR, inside Isabelle [2]. Webb et al. [2] describes that there are 2 proof obligations (or subgoals) 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 for the optimization;
2) is obviously true – a proof can be found;
3) would require manual proving by *"proof experts"*.

To leverage Isabelle, VeriTest utilizes Isabelle Client - Server [6] interactions as a method of passing optimization rules to prove inside Isabelle. Isabelle Server acts as the core Isabelle process that allows theorems and all the required facts to be loaded up and processed in parallel (See Section IV-C) by Isabelle/ML which is ideal for the nature of VeriTest [6, Ch. 4.2.6]. Due to the fact that theorems can interfere with one another, VeriTest ensures that each theorem are mutually exclusive by separating the sessions that they are run on, making them essentially stateless.

Figure 1 demonstrates the solution architecture of VeriTest. As there are multiple ways of interacting with Isabelle Server [6, Ch. 4.2], VeriTest abstracts that 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 and proven automated theorem prover. Passing optimization rules into Isabelle Client is done in parallel, utilizing asynchronous functions managed by a thread pool inside the system. These asynchronous functions implements a circuit-breaker pattern when either a proof or a counterexample is found in order to provide fast feedback.

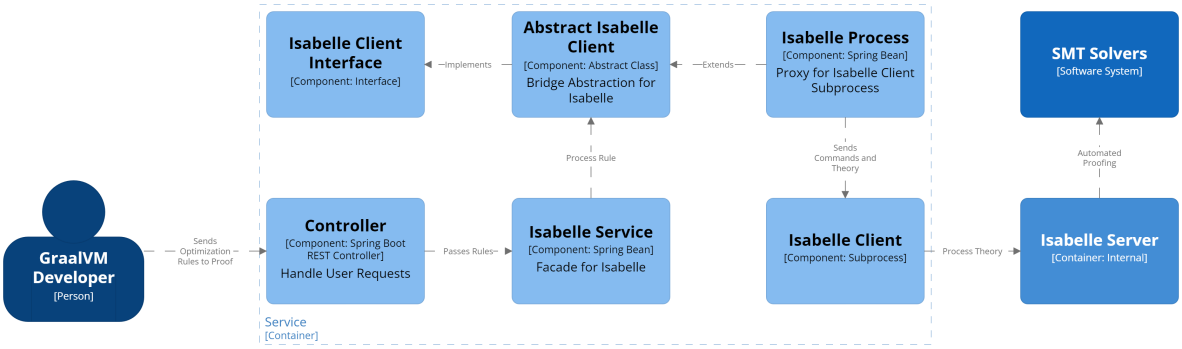

Figure 1. Solution Architecture of VeriTest

A. Finding Counterexample

In order to find counterexamples, we utilize Nitpick and Quickcheck from Isabelle. Optimization expression is 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].

B. Finding Proof

Sledgehammer is invoked recursively over the course of verifying the optimization rule. As after Sledgehammer is invoked, there may be multiple possible proofs that is available, and it may not completely prove all proof obligations.

C. Limitations

Over the course of the implementation, we found a key flaw inside Isabelle Server [6]. Unexpectedly, 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 seem to support concurrent use.

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 existing Veriopt optimization rules based on runtime (s)

Evaluation is done by exporting the existing Veriopt's expression canonicalization optimization rules and iterating it 5 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 demonstrated consistency over automatic proofing, finding counterexamples, and finding malformed rules. This is due to the rules being processed inside Isabelle/ML, instead of passed on to an external SMT solver. The main bottleneck of the process is due to the limitations of Isabelle Server (See Section IV-C) and the stateless manner of which it is processed.

Furthermore, we found that VeriTest is capable of utilizing existing lemmas inside Veriopt's current working to find proofs for the optimization rule. The discrepancy of the runtime is due to the fact that some optimization rules are inherently more complex than others (e.g., *XorIsEqual_64_1* and *XorIsEqual_64_4*).

Surprisingly, some of the existing optimization rules are malformed (e.g., *AbsIdempotence* and *AbsNegate*). 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. Lecture Notes in Computer Science, Z. Hou and V. Ganesh, Eds., vol. 12971. Cham: Springer International Publishing, 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*, ser. CPP 2023. New York, NY, USA: Association for Computing Machinery, 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*, C. Tinelli and V. Sofronie-Stokkermans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6989, pp. 12–27, series Title: Lecture Notes in Computer Science.

[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. Cham: Springer International Publishing, 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