Extensible Verified Validation for LLVM Optimizations

(poster only submission)

Sungkeun Cho[1⋆], Joonwon Choi[2⋆], Jeehoon Kang[1⋆], Chung-Kil Hur[1], and Kwangkeun Yi[1]

[1] Seoul National University, Korea
[2] Massachusetts Institute of Technology, USA

1 Challenge: Verification of Mainstream Compiler

Since compilers are indispensable tools for software development, their reliability is crucial, especially for safety-critical software. However, as recent researches [4, 1] have shown, mainstream C compilers such as GCC and LLVM contain plenty of bugs that matter practically. For example, [4] found 79 bugs in GCC and 202 bugs in LLVM, and [1] found 79 bugs in GCC and 68 bugs in LLVM.

Recent advances in theorem proving technology have made it possible to write a completely bug-free compiler such as CompCert [2], yet this technology have not been fully applied to mainstream C compilers. This is due to the huge amount of work required; for example, LLVM is more than two million lines of code in C++.

Thus, our goal is to develop a verified validation technique that can scale well to handle mainstream compilers such as LLVM. For this, we combine ideas from existing approaches, which we summarize as follows.

- *Translation validation* approach [3] observes that many different optimizations can be validated in the same way and developed several "automatic" validation algorithms. However, since they are fully automatic tools, they are limited in finding complicated invariants for validation. For example, they have not been applied to the instruction combine pass; for e.g. an optimization that exploits the commutativity of integer addition.
- Existing *verified validation* approach [2] is to develop and verify a validator that is specialized to one particular optimizer. It can be *complete* (*i.e.*, it can find invariants for validation of the optimizer) since the invariants are "manually" specified. Since verifying each validator requires a significant amount of work, it is practically impossible to cover hundreds of different optimizers this way due to the huge amount of work required.

2 Our Approach: Extensible Verified Validation

To address the challenge, we propose an extensible verified validation approach, which takes advantages of the two existing approaches. First, like the existing

⋆ The first three authors have equal contribution to this work.

verified vadliation approach, we manually specify invariants for each optimization in order to make our approach complete. However, to reduce the verification effort, we also take the idea of translation validation that many different optimizers can be validated in the same way. Thus, we design a validator that share the main structure of the reasoning (**generality**) but allows each optimizer to have its own hints and inference rules (**extensibility**), so that it can be applied to many different optimizers, but minimize the amount of work for verification.

As a first step to show feasibility of our approach, we have developed an extensible verified validator and applied it to micro-optimizations (about 400 in total) in the instruction combine pass of the LLVM compiler. Our validator currently covers more than 100 micro-optimizations, with machine-checked correctness proof in Coq.

For **generality**, our validator is based on a simple form of relational Hoare reasoning. More specifically, we give an invariant to each line of source code, and the validator checks if the invariant holds at each step. We expect that Hoare-style reasoning is general enough to cover all optimizations of LLVM compiler that do not change CFG (control flow graph).

For **extensibility**, one can freely add to the validator user-defined inference rules (that may be specific to a particular optimization) so long as the correctness of the rules is provided. To utilize newly added inference rules, hint generation code is inserted to the optimizer, and generates hints that indicate which inference rules are used in which lines of source code.

As claimed above, our validator scales well to cover many different optimizations. The key point is that we need to verify the shared main structure of our validator just once. Indeed, it is this main structure that requires a significant amount of work for verification. Verification of each inference rule is usually very simple. For example, the rule for commutativity of integear addtion just requires to prove that the commutativity holds.

3 Future Work

We will extend our validator to cover most of the LLVM optimization level 1 (`-O1`). We believe that Hoare-style reasoning is general enough not only to cover instruction combine pass, but also to cover all structure-preserving optimization passes (110 out of 124) of LLVM `-O1`. Developing different validators for the remaining 14 passes is also interesting future work.

References

1. V. Le, M. Afshari, and Z. Su. Compiler validation via equivalence modulo inputs. In *PLDI*, 2014.
2. X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.
3. G. C. Necula. Translation validation for an optimizing compiler. In *PLDI*, 2000.
4. X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in C compilers. In *PLDI*, 2011.