Leveraging LLVM Optimizations to Speed up Constraint Solving

Satisfiability Modulo Theories (SMT) constraints are first-order logical formulas used to encode program analysis problems. SMT solving is the foundation for LLVM-based analysis tools such as KLEE [2], and analyses of the LLVM project itself, such as Alive and its progeny [4–6]. When solvers can handle problems more quickly, these tools perform better. For example, reducing solving time equates to greater code coverage in symbolic execution of LLVM IR programs [2]. State-of-the-art solvers for SMT constraints, including CVC5 [1] and Z3 [3] efficiently reason about some problems over bitvectors and floating-point numbers, yet many constraints still take a prohibitively long time to solve. We take a fresh perspective on speeding up SMT solving: by harnessing the power of existing LLVM optimization passes, we can simplify constraints before passing them to a solver. Thus while existing work uses constraint solving to analyze programs, we "flip the script" and give solver users the benefit of the extensive effort made to develop LLVM optimization passes. Our strategy is to translate SMT constraints into LLVM IR, apply the optimizer, and then translate back. This alleviates manual developer effort in understanding both solver and LLVM internals, and can apply to any SMT solver since it pre-processes constraints.

We instantiate our approach with a practical tool called SLOT (SMT-LLVM Optimizing Translation) [7], which takes as input SMT constraints and produces simplified constraints, which are then fed to a solver. SLOT consists of three components: a *frontend* which converts SMT constraints to IR, LLVM's existing compiler optimizer, and a *backend*, which translates IR functions back into SMT constraints. The key challenge for SLOT is bridging the semantic gap between SMT constraints and LLVM IR. While many SMT-LIB functions have direct equivalents in LLVM (bitvector addition, or floating–point division, for example), the languages differ in subtle ways. For instance, bitvector division in LLVM is undefined for some inputs on which it is defined in SMT-LIB. SMT-LIB is missing definitions of several critical LLVM bit operation intrinsics such as counting set bits, while LLVM is missing some SMT-LIB operations like `bvsmod`. We bridge the gap by developing a one-to-one mapping between SMT-LIB function applications and sequences of LLVM instructions.

Our extensive empirical evaluation on more than 100,000 benchmarks from the standard SMT-LIB benchmark set demonstrates that SLOT allows LLVM optimizations to substantially speed up SMT solving, especially for complex constraints which would otherwise take a long time to solve. SLOT speeds up average solving time by more than 2× for bitvector and floating–point constraints, and as high as 3× for mixed constraints. It also increases the number of solvable constraints at fixed timeouts by up to 80%. Moreover, we expand the uses of LLVM's optimization repertoire by giving solver developers new insights about which existing compiler strategies could be useful in the solving context—we find that simple peephole optimizations, reassociation, and global value numbering are the most effective at speeding up solving. Our results therefore expand the uses of LLVM to SMT solvers which play a key role in both industry and academia, and complete the circle by improving analysis tools useful to LLVM's further development.

References

[1] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysfis of Systems - 28th International Conference, TACAS*. 415–442.

[2] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI*. 209–224.

[3] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS*. 337–340.

[4] Nuno P. Lopes, Juneyoung Lee, Chung-Kil Hur, Zhengyang Liu, and John Regehr. 2021. Alive2: bounded translation validation for LLVM. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 65–79.

[5] Nuno P. Lopes, David Menendez, Santosh Nagarakatte, and John Regehr. 2015. Provably correct peephole optimizations with alive. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 22–32.

[6] David Menendez, Santosh Nagarakatte, and Aarti Gupta. 2016. Alive-FP: Automated Verification of Floating Point Based Peephole Optimizations in LLVM. In *Static Analysis - 23rd International Symposium, SAS*. 317–337.

[7] Benjamin Mikek and Qirun Zhang. 2023. Speeding up SMT Solving via Compiler Optimization. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023 (to appear)*.