Faster SMT Solving via Constraint Transformation

Benjamin Mikek
bmikek@gatech.edu
Georgia Institute of Technology
USA

Qirun Zhang
qrzhang@gatech.edu
Georgia Institute of Technology
USA

Satisfiability Modulo Theories (SMT) constraints are first-order logical formulas with functions and variables from various theories, such as real numbers, integers, bitvectors and floating-point numbers. Many program analysis tools generate SMT constraints over the bounded theories of bitvectors and floating-point numbers. SMT solving of these theories is therefore central to program analysis applications such as symbolic execution [8, 14], program synthesis [7, 12], and program verification [5, 13]. The unbounded theories of real numbers and integers provide the foundation for yet more tools. For example, real number constraints are useful in modeling automata [9], while integer arithmetic has been used to encode the constant multiplication problem [17] and in developing better answer set programming techniques [22].

The performance of practical tools is directly tied to the performance of solvers; when solvers can handle problems quickly, the tools perform better. For example, in symbolic execution, reducing solving time equates to greater code coverage [8]. State-of-the-art solvers for SMT constraints, including CVC5 [2] and Z3 [11], use a complex mix of heuristics, theory-specific engines, and SAT solver calls to efficiently reason about a problem. Yet many constraints still take a prohibitively long time to solve, reducing the usefulness of solver-supported tools to end users.

The most popular approach to speeding up SMT solving has been to embed new and more powerful strategies within solvers. These have sometimes taken the form of new solvers like Boolector [19], or of new algorithms in existing solvers. For example, Berzish *et al.* [4] introduce new heuristics for string constraints involving regular expressions while Bjørner *et al.* [6] improve Z3's performance for custom theories. Reynolds *et al.* develop many optimizing rewriting rules [20] for the bitvector theory, which improves the performance of counterexample-guided solving [21]. FastSMT [1] speeds up solving by using machine learning to select solver heuristics.

We propose a new perspective on speeding up SMT solving: instead of focusing on solver internals, we turn our attention to constraints. Our key insight is that pre-processing constraints before passing them to a solver can simplify the constraints, reduce the solver's workload, and thereby speedup solving. Our strategy has three conceptual advantages. First it allows solver users to benefit from better performance without detailed knowledge of solver internals. Second, since our approach pre-processes constraints, it can be used with any solver(s). Third, it allows the introduction of simplification strategies originating outside the SMT domain.

SMT-LLVM Optimizing Translation

In existing work [16], we instantiate the pre-processing approach and show its usefulness by harnessing the power of compiler optimization. Our insight is to repurpose existing compiler optimization techniques to the SMT problem. In particular, we develop **S**MT-**L**LVM **O**ptimizing **T**ranslation (SLOT), which can directly optimize input SMT-LIB formulas.

While most constraint optimization strategies have focused on transforming SMT formulas themselves, SLOT bypasses the need to implement simplifications within a solver by translating the constraints into LLVM IR and then applying LLVM's existing optimization passes. While not all compiler optimizations are useful for the SMT context, the combination of SLOT with existing solvers creates a sieve: some constraints are caught quickly by existing solver heuristics, while others are handled better by SLOT.

We have implemented SLOT for the SMT theories of bitvectors and floating-point numbers. Constraints in these theories are the most relevant to software engineering because they model machine arithmetic; for example, they are used in practical tools for symbolic execution [8], translation validation [15], and program synthesis [7]. In addition, we provide proof that the semantics of these two theories can be exactly represented in LLVM IR. The key challenge for SLOT is bridging the substantial semantic gap between SMT constraints and LLVM IR. Translation is particularly challenging because the languages, one declarative and the other imperative, were designed for entirely different purposes.

As shown in Figure 1, SLOT consists of three components: a *frontend* which converts SMT constraints to LLVM IR, the *optimizer*, which optimizes IR code, and a *backend*, which translates IR functions back into SMT constraints. While many SMT-LIB functions have direct equivalents in LLVM (bitvector addition, or floating–point division, for example), the semantics of 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.

We have evaluated our translation and optmization strategy on more than 100,000 benchmarks from the SMT-LIB benchmark set—those for bitvectors, floating-point numbers, and their combination [3]. SLOT is able to produce output constraints which are equisatisfiable to the original for every benchmark which does not time out, empirically validating our proofs of the correctness of frontend and backend translation. Since the purpose of compiler optimizations is to simplify instructions, not constraint solving, SLOT may produce more complex constraints on some inputs. However, such cases are rare and their impact can be combatted by running SLOT according to portfolio methodology [24].

Our empirical evaluation demonstrates that SLOT can substantially speed up SMT solving, especially for complex constraints which would otherwise take a long time to solve. Our approach increases the number of solvable constraints at fixed timeouts by up to 18% for bitvectors, 14% for floating-point numbers, and 80% for mixed benchmarks. Moreover, SLOT can solve constraints for

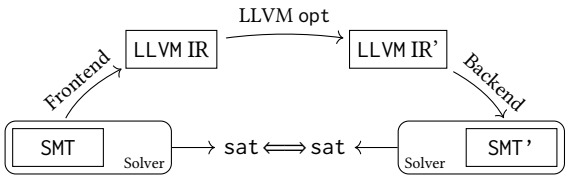

Figure 1: Overview of SLOT's translation and optimization process. The output constraint (SMT') is satisfiable if and only if the original constraint (SMT) is satisfiable.

which all tested solvers time out. We also observe mean speedups above 2× for bitvector and floating–point constraints, and as high as 3× for mixed constraints. By measuring which optimization passes contribute to the speedup, we find that simple peephole optimizations, reassociation, and global value numbering are the most effective at speeding up solving. This result provides insight into what optimizations are left on the table by existing solvers.

The practical results for translation and optimization with LLVM also demonstrate something general about the pre-processing approach: performance gains increase with original solving time because for very small constraints, the overhead of pre-processing may exceed the cost of simply running a solver. For example, one benchmark using SLOT and Z3 is sped up from 0.06 seconds to 0.02 seconds, but SLOT takes 0.24 seconds to translate and optimize it, a substantial proportional, if not absolute, slowdown. The performance decrease for small and simple constraints is offset by large performance gains for complex constraints.

Unbounded Theories

The pre-processing strategy, however, is not only limited to translation and optimization with LLVM; we also consider how to transform constraints from unbounded theories into bounded ones. This type of transformation is of interest because, although SMT solvers handle unbounded values in the theories of integer and real arithmetic, they are implemented in traditional programming languages, where all datatypes are bounded. The gap between the requirements of the theory and the practical reality of implementation makes solving unbounded constraints challenging and expensive.

Unlike the theories of bitvectors and floating-point numbers, the search space for unbounded theories is infinite on its face. For constraints on real numbers, satisfiability is at least decidable, but no theoretical bounds exist [23]. For integer constraints, the picture is no better. *Linear* integer constraints (without multiplication or division of variables) are decidable and satisfying assignments of variables for these constraints are bounded [18]. Unfortunately, the bounds on solutions are exponential in the number of assertions (inequalities), meaning that they become impractically large even for relatively small numbers of assertions. Useful SMT problem instances can run to thousands of inequalities. For *nonlinear* integer arithmetic, not only are solutions unbounded, but the question of satisfiability is undecidable [10]. Both the theoretical complexity of the problem and the engineering issues faced by solver developers make it difficult to improve performance for unbounded theories.

We propose a theory arbitrage to overcome the challenges of solving constraints in unbounded theories. Our key insight is that unbounded real numbers and integers each have a bounded counterpart SMT theory: floating-point numbers and bitvectors. Our strategy is to use constraints in the bounded theories to solve unbounded constraints. We begin with an original constraint S in an unbounded theory, and convert to constraint S' with the same semantics, but in a bounded theory. The benefit of our approach is that it speeds up solving by taking advantage of solvers' superior performance on bounded theories. In our experiments for example, we find that Z3 can take up to 5× longer to solve a nonlinear integer constraint than a similar bitvector one.

The main challenge in effecting this transformation is to achieve the translation without changing the semantics of the underlying constraint. In particular, bounded SMT sorts inherently cannot represent all values in an unbounded theory, either because they are too large (integers and real numbers), or because they are too precise (real numbers). We address this challenge by first analyzing the problem to reason about which bounds are likely to be correct and, second, introducing an underapproximation that guarantees correctness in a subset of cases. These two fundamental steps give rise to a framework for converting unbounded SMT constraints into bounded ones, which is adaptable and efficient.

In our framework, we achieve the first step by introducing a novel and general abstract interpretation strategy to infer bounds on the original SMT problem, giving us the tools to convert S into S'. This strategy is adaptable to the different notions of boundedness in integer and real number problems. However, the theoretical properties of the problem mean that the inferred bounds are not always sufficient. Thus, in the second step, we introduce underapproximation. If the new constraint S' is unsatisfiable, we revert to the original constraint, providing no performance improvement and guaranteeing the correct satisfiability result. If S' is satisfiable, we verify that its satisfying assignment also satisfies S before returning a result; in the typical case, this process is faster than directly solving S. In aggregate, this underapproximation strategy guarantees correctness while allowing users to benefit from increased performance for many constraints.

Our preliminary experimental results for transformation from unbounded theories to bounded ones are encouraging. For some constraints the transformation achieves huge speedups, converting constraints that take hundreds of seconds in unbounded form to a new form which takes under one second to solve. However, other constraints become harder for solvers to deal with. Our empirical results suggest that the speedup is the greatest for nonlinear integer constraints, with only very small changes for linear real constraints.

Conclusion

In summary, we have presented a general framework for speeding up SMT solving: instead of improving solver internals, we aim to transform and simplify constraints. We have instantiated this strategy in two ways and find encouraging concrete results. Our main contributions are the following:

- We propose a fresh perspective on speeding up SMT solving by transforming constraints rather than adding to solvers.
- We develop a constraint translation strategy which uses compiler optimizations to speed up SMT solving, achieving substantial speedups in practice.
- We create an under-approximative transformation from unbounded SMT theories to bounded ones, achieving large speedups for nonlinear integer constraints.

References

[1] Mislav Balunovic, Pavol Bielik, and Martin T. Vechev. 2018. Learning to Solve SMT Formulas. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS)*. 10338–10349.

[2] 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.

[3] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2017. *The SMT-LIB Standard: Version 2.6*. Technical Report. Department of Computer Science, The University of Iowa. Available at `www.SMT-LIB.org`.

[4] Murphy Berzish, Mitja Kulczynski, Federico Mora, Florin Manea, Joel D. Day, Dirk Nowotka, and Vijay Ganesh. 2021. An SMT Solver for Regular Expressions and Linear Arithmetic over String Length. In *Computer Aided Verification - 33rd International Conference, CAV*. 289–312.

[5] Dirk Beyer, Matthias Dangl, and Philipp Wendler. 2018. A Unifying View on SMT-Based Software Verification. *J. Autom. Reason.* 60, 3 (2018), 299–335.

[6] Nikolaj Bjørner, Clemens Eisenhofer, and Laura Kovács. 2023. Satisfiability Modulo Custom Theories in Z3. In *Verification, Model Checking, and Abstract Interpretation, VMCAI*. 91–105.

[7] Tim Blazytko, Moritz Contag, Cornelius Aschermann, and Thorsten Holz. 2017. Syntia: Synthesizing the Semantics of Obfuscated Code. In *26th USENIX Security Symposium, USENIX Security*. 643–659.

[8] 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.

[9] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. 2012. A quantifier-free SMT encoding of non-linear hybrid automata. In *Formal Methods in Computer-Aided Design, FMCAD 2012*. IEEE, 187–195.

[10] Martin Davis, Yuri Matijasevič, and Julia Robinson. 1976. Hilbert's tenth problem. Diophantine equations: positive aspects of a negative solution. *American Math. Soc Providence* 1 (1976).

[11] 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.

[12] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE 2010*. ACM, 215–224.

[13] Juneyoung Lee, Dongjoo Kim, Chung-Kil Hur, and Nuno P. Lopes. 2021. An SMT Encoding of LLVM's Memory Model for Bounded Translation Validation. In *Computer Aided Verification - 33rd International Conference, CAV*. 752–776.

[14] Daniel Liew, Daniel Schemmel, Cristian Cadar, Alastair F. Donaldson, Rafael Zähl, and Klaus Wehrle. 2017. Floating-point symbolic execution: a case study in n-version programming. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE*. 601–612.

[15] 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.

[16] 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)*.

[17] V. Manquinho J. Monteiro N. P. Lopes, L. Aksoy. 2010. *Optimally Solving the MCM Problem Using Pseudo-Boolean Satisfiability*. Technical Report RT/43/2010. INESC-ID.

[18] Christos H. Papadimitriou. 1981. On the Complexity of Integer Programming. *J. ACM* 28, 4 (oct 1981), 765–768.

[19] Mathias Preiner, Aina Niemetz, and Armin Biere. 2017. Counterexample-Guided Model Synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS*. 264–280.

[20] Andrew Reynolds, Haniel Barbosa, Cesare Tinelli, Aina Niemetz, Andres Noetzli, Mathias Preiner, and Clark Barrett. 2018. Rewrites for SMT Solvers Using Syntax-Guided Enumeration. http://homepage.divms.uiowa.edu/~ajreynol/pres-smt2018.pdf

[21] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett. 2015. Counterexample-Guided Quantifier Instantiation for Synthesis in SMT. In *Computer Aided Verification - 27th International Conference, CAV 2015*. 198–216.

[22] Da Shen and Yuliya Lierler. 2018. SMT-Based Constraint Answer Set Solver EZSMT+ for Non-Tight Programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018*. 67–71.

[23] Alfred Tarski. 1998. A Decision Method for Elementary Algebra and Geometry. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer Vienna, 24–84.

[24] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. 2009. A Concurrent Portfolio Approach to SMT Solving. In *Computer Aided Verification, 21st International Conference, CAV*. 715–720.