The solsmt Solver in SMT-COMP 2022

Christian Reitwießner, Mate Soos
Ethereum Foundation

July 1, 2022

1 Overview

The code optimizer of the Solidity compiler used by the vast majority of contracts deployed on the Ethereum Blockchain [But13] tries to remove redundant conditions to optimize the execution of the contracts running on the Ethereum Virtual Machine (EVM). In order to maintain perfect reproducibility and reduce the bug attack surface, the compiler implements its own simplistic SMT solver, *solsmt*, which also exposes an SMTLib2 interface.

The solver, as part of the compiler, is written in C++ and can only handle quantifier-free linear rational arithmetic (QF_LRA). It is designed for incremental queries as would be expected when analyzing intermediate language code.

The main website of the compiler is `https://soliditylang.org` and its source code can be found at `https://github.com/ethereum/solidity/`.

2 Components of solsmt

The satisfiability part of the solver is CDCL-based, inspired by MiniSat 1.14 [ES03], and the linear rational solver is based on the ideas from the article "A Fast Linear-Arithmetic Solver for DPLL(T)" [DdM06].

To the best of our knowledge, the main theoretical innovation of the tool is that the theory solver stores the rational problem split into sub-problems concerning minimal disjoint sets of variables. Specifically, the solver only allows adding constraints and not removing them. As also done by other systems, a stack of solvers is maintained so that the CDCL routine can backtrack to previous states.

Whenever a constraint is added, all sub-problems the new constraint uses variables from are combined into one sub-problem. Or if none of the variables appear in any of the existing sub-problems, a new sub-problem is created. Then the constraint is added to the new or combined sub-problem.

This way, the problems to solve always remain small and do not need to be re-solved if they did not change between two calls to the theory solver.

The CDCL part of *solsmt* uses geometric restarts, does not delete learnt clauses, and does not pre- or inprocess the instance. Clause learning is done via the classic first UIP system as pioneered by Chaff [MMZ+01], without any form of clause minimization. For the decision heuristic, we use VSIDS as per MiniSat, and phase saving as per [PD07]. In other words, the CDCL part of the solver is extremely minimalistic in order to keep code complexity low.

References

[But13] Vitalik Buterin. Ethereum white paper: A next generation smart contract and decentralized application platform. `https://ethereum.org/en/whitepaper/`, 2013.

[DdM06] Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In João Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.