# The OpenSMT Solver in SMT-COMP 2021 – To Be Updated

Martin Blicha, Antti E. J. Hyvärinen, Matteo Marescotti, and Natasha Sharygina

Università della Svizzera italiana (USI), Lugano, Switzerland

## 1 Overview

OpenSMT [8] is a T-DPLL based SMT solver [13] that has been developed at USI, Switzerland, since 2008. The solver is written in `C++` and currently supports the quantifier-free logics of equality with uninterpreted functions (QF_UF), and linear real arithmetic (QF_LRA). The solver has a rudimentary support for quantifier-free linear integer arithmetic (QF_LIA) based on branch-and-bound, and supports some aspects of bit-vector logic (QF_BV).

In comparison to 2019, the 2020 competition entry features a wide range of performance improvements especially in our implementation of the Simplex algorithm [6], and to lesser extent in simplification, and in the Egraph algorithm [5]. Additional improvement regards the performance of incremental solving. Finally, the support for producing model for satisfiable instances has been added for QF_LRA. In the process, the solver high-level architecture improved, a few bugs related to solver soundness has been fixed, and low-level code cleaning resulted in elimination of standard compiler warnings.

The most important optimization added in the last year is a decision heuristic in SAT solver for deciding the polarity of the chosen decision variable based on suggestion from the theory solver. Choosing polarity that is consistent with the current satisfying assignment of the theory solver yielded a huge performance improvement on QF_LRA benchmarks and a small improvement on QF_UF benchmarks. Other important optimizations in LRA theory solver include lazy tableau representation and a single backup assignment.

OpenSMT features not exercised in the competition include support for a wide range of interpolation algorithms for propositional logic [2], linear real arithmetic [4], and uninterpreted functions [3] (now available also in the incremental mode); an experimental lookahead-based search algorithm [9] as an alternative to the more standard CDCL algorithm; and features that support search-space partitioning in particular designed for parallel solving [10].

## 2 External Code and Contributors

The SAT solver driving OpenSMT is based on the MiniSAT solver [7], and the rational number implementation is inspired by a library written by David Monniaux. Several people have directly contributed to the OpenSMT code. In alphabetical order, the major contributors are Leonardo Alt (Ethereum Foundation), Sepideh Asadi (USI), Martin Blicha (USI, Charles University), Roberto Bruttomesso (Cybersecurity / Nozomi Networks), Antti E. J. Hyvärinen (USI), Matteo Marescotti (USI), Rodrigo Benedito Otoni (USI), Edgar Pek (University of Illinois, Urbana-Champaign), Simone Fulvio Rollini (United Technologies Research Center), Parvin Sadigova (King's College London), and Aliaksei Tsitovich (Sonova). The solver is being developed in Natasha Sharygina's software verification group at USI.

# 3 Utilization

OpenSMT is used in a range of projects as a back-end solver. Recent examples include its use as an interpolation engine of the Sally model checker [11] which won the first and the second place in the transition systems category in the constrained Horn clause competition 2019 and 2020, respectively. OpenSMT also forms the basis of our own model checkers such as HiFrog [1]. OpenSMT is compatible with the SMTS parallelization framework [12].

# 4 Availability

The source code repository and more information on the solver is available at

- `https://github.com/usi-verification-and-security/opensmt` and
- `http://verify.inf.usi.ch/opensmt`

# References

[1] Leonardo Alt, Sepideh Asadi, Hana Chockler, Karine Even-Mendoza, Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina. HiFrog: SMT-based function summarization for software verification. In *Proc. TACAS 2017*, volume 10206 of *LNCS*, pages 207–213. Springer, 2017.

[2] Leonardo Alt, Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina. A proof-sensitive approach for small propositional interpolants. In *Proc. VSTTE 2015*, volume 9593 of *LNCS*, pages 1–18. Springer, 2016.

[3] Leonardo Alt, Antti Eero Johannes Hyvärinen, Sepideh Asadi, and Natasha Sharygina. Duality-based interpolation for quantifier-free equalities and uninterpreted functions. In *Proc. FMCAD 2017*, pages 39–46. IEEE, 2017.

[4] Martin Blicha, Antti E. J. Hyvärinen, Jan Kofron, and Natasha Sharygina. Decomposing Farkas interpolants. In *Proc. TACAS 2019*, volume 11427 of *LNCS*, pages 3–20. Springer, 2019.

[5] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[6] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proc. CAV 2006*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006.

[7] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. SAT 2004*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.

[8] Antti E. J. Hyvärinen, Matteo Marescotti, Leonardo Alt, and Natasha Sharygina. OpenSMT2: An SMT solver for multi-core and cloud computing. In *Proc. SAT 2016*, volume 9710 of *LNCS*, pages 547–553. Springer, 2016.

[9] Antti E. J. Hyvärinen, Matteo Marescotti, Parvin Sadigova, Hana Chockler, and Natasha Sharygina. Lookahead-based SMT solving. In *Proc. LPAR-22*, volume 57 of *EPiC Series in Computing*, pages 418–434. EasyChair, 2018.

[10] Antti E. J. Hyvärinen, Matteo Marescotti, and Natasha Sharygina. Search-space partitioning for parallelizing SMT solvers. In *Proc. SAT 2015*, volume 9340 of *LNCS*, pages 369–386. Springer, 2015.

[11] Dejan Jovanovic and Bruno Dutertre. Property-directed k-induction. In *Proc. FMCAD 2016*, pages 85–92. IEEE, 2016.

[12] Matteo Marescotti, Antti E. J. Hyvärinen, and Natasha Sharygina. SMTS: distributed, visualized constraint solving. In *Proc. LPAR-22*, volume 57 of *EPiC Series in Computing*, pages 534–542. EasyChair, 2018.

[13] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937 – 977, 2006.