The OpenSMT Solver in SMT-COMP 2022

Masoud Asadzade, Martin Blicha, Antti E. J. Hyvärinen, Rodrigo Otoni, and Natasha Sharygina

Università della Svizzera italiana (USI), Lugano, Switzerland

1 Overview

OpenSMT [5] is a T-DPLL based SMT solver [9] 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), arrays with extensionality (QF_AX), linear real and integer arithmetic (QF_LRA, QF_LIA) and their combinations with uninterpreted functions (QF_UFLRA, QF_UFLIA), and real and integer difference logics (QF_RDL, QF_IDL). OpenSMT also supports some aspects of bit-vector logic (QF_BV).

In comparison to 2021, the 2022 competition entry supports for the first time the combinations of arithmetic with uninterpreted functions and the theory of arrays. The solver's performance has continued to improve over the season. Notable changes include the use of phase saving, switching to glue heuristic for learned clauses, and using the Luby restart scheme in the SAT solver, and avoiding repeated memory allocation and freeing by implementing pooling for our rational numbers. The SAT solver improvements were suggested by Mate Soos and the pooling was implemented by @MicTarHal.

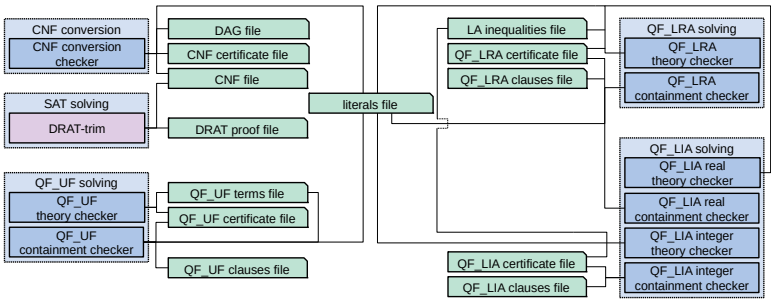
We participate in the proof track of the competition with an approach that is based on producing traces of the solver [10], although this feature has not yet been merged to the main repository.

Other notable features in the developer's perspective include our gradual switch to the C++-17 standard and the support for static builds (in Linux).

2 Cloud and Parallel Solver

The parallel version of OpenSMT, called SMTS, runs on our parallelization infrastructure described in [8]. The system is a client-server architecture that communicates with a custom protocol over TCP/IP. New features for 2022 include the complete implementation of the *partition tree protocol* with clause sharing. On a high level, the parallel solver partitions the instance dynamically on-demand and allows clauses to be shared between solvers working on different instances whenever this is allowed based on the information in the solvers. The dynamic partitioning is implemented as an iteration in a tree of consisting of partitioned instances, and the instances in inner nodes are considered equally to those in the leaves. This aproach guarantees under reasonable assumptions on the instance's run time distribution that the parallel solver will not be slower than the sequential solver [6]. For a fair and load-balanced scheduling of solving of the partitions, the solvers are distributed over the instances on parameters such as whether the instance has already been attempted, how many solvers are working on the instance, and how deep in the tree the instance is.

We support running the solver in the in AWS infrastructure, but it can also be run, e.g., in multi-core environments. We have entered two versions of the solver to the cloud and parallel tracks: *SMTS portfolio* which randomises the SAT solver by choosing 2% of the decision vari-



ables randomly; and *SMTS cube-and-conquer*, which uses the *parallelization tree* [6] approach to implement a version of search-space-partitioning.

3 External Code and Contributors

The SAT solver driving OpenSMT is based on MiniSAT [4], 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), Masoud Asadzade (USI), Martin Blicha (USI, Charles University), Konstantin Britikov (USI), Roberto Bruttomesso (Cybersecurity / Nozomi Networks), Antti E. J. Hyvärinen (USI), Václav Luňák (Charles University), Matteo Marescotti (USI), MicTarHal, Rodrigo Benedito Otoni (USI), Edgar Pek (University of Illinois, Urbana-Champaign), Simone Fulvio Rollini (United Technologies Research Center), Parvin Sadigova (King's College London), Mate Soos (Ethereum Foundation) Andrew Jones (Vector), and Aliaksei Tsitovich (Sonova). The solver is being developed in Natasha Sharygina's software verification group at USI.

4 Utilization

OpenSMT is used in a range of projects as a back-end solver. It is the basis for the CHC solver Golem [3] which won the first place in LIA-lin, LIA-nonlin, LRA-TS and LRA-TS-parallel CHC-COMP 2021. It has been used as an interpolation engine of the Sally model checker [7] which won the first and the second place in the transition systems category in the constrained Horn clause competition 2019 and 2020, respectively. OpenSMT is also the basis of the model checkers HiFrog [1] and UpProver [2].

5 OpenSMT's Trail Format

OpenSMT is able to produce trails that act as guarantees for the correctness of its executions. In this sense they are similar to proofs in first-order logic. The detailed description of the trails is in [10], and Fig. 5 gives an overview of how the trails are organised. In essence, the trails consist of the *DRAT* trail of the SAT solver, trails for the Tseitin transformation, and diverse trails for certifying the theory-specific reasoning and the related clauses.

To be able to demonstrate the trail format in SMT-COMP's new proof demonstration track, we implemented a system that is able to present the trails as an s-expression. The

approach is straightfoward: the directory structure is compressed using standard unix utilities, the compressed file is transformed into printable form using a base64-encoding, and then printed in an smt-lib-inspired s-expression syntax.

6 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] Sepideh Asadi, Martin Blicha, Antti E. J. Hyvärinen, Grigory Fedyukovich, and Natasha Sharygina. Incremental verification by smt-based summary repair. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 77–82. IEEE, 2020.

[3] Martin Blicha, Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina. Transition power abstractions for deep counterexample detection. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 524–542. Springer, 2022.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] Rodrigo Otoni, Martin Blicha, Patrick Eugster, Antti E. J. Hyvärinen, and Natasha Sharygina. Theory-specific proof steps witnessing correctness of SMT executions. In *Proc. DAC 2021*. IEEE, 2021. To appear.