CVC4 at the SMT Competition 2020

Clark Barrett[1], Haniel Barbosa[3], Martin Brain[4], Ahmed Irfan[1], Makai Mann[1], Mudathir Mohamed[2], Aina Niemetz[1], Andres Nötzli[1], Alex Ozdemir[1], Mathias Preiner[1], Andrew Reynolds[2], Ying Sheng[1], Cesare Tinelli[2], and Amalee Wilson[1]

[1]Stanford University
[2]The University of Iowa
[3]Universidade Federal de Minas Gerais
[4]University of Oxford and City, University of London

***Abstract*—This paper is a description of the CVC4 SMT solver as entered into the 2020 SMT Competition. We only list important differences from the 2019 SMT Competition version of CVC4. For further and more detailed information about CVC4, please refer to the original paper [13], the CVC4 website [10], or the source code on GitHub [9].**

Overview

CVC4 is an efficient open-source automatic theorem prover for SMT problems. It can be used to prove the validity (or, dually, the satisfiability) of first-order formulas in a large number of built-in logical theories and combinations thereof.

CVC4 is intended to be an open and extensible SMT engine, and it can be used as a stand-alone tool or as a library, with essentially no limit on its use for research or commercial purposes (see the section on its license below for more information).

New Features and Improvements

The CVC4 configuration entered in the SMT Competition 2020 is an improved and extended version of the version that entered SMT-COMP 2019. Most notably, it features the following extensions.

String Solver: The string solver has been significantly improved since last year's submission. Some of the major improvements include:

- More aggressive context-dependent simplifications (extending previous work [20])
- Better reductions of extended functions (fine-tuning and sharing of witnesses)
- Lazy processing of regular expression intersections

For conversions between strings and numbers, we implement a novel method that does not split on the character values in the string [19].

Eager Bit-Blasting Solver: We use CaDiCaL [1, 14] version 1.2.1 as back end for the eager bit-blasting engine, both for non-incremental and incremental solving. For non-incremental `QF_UFBV`, we enable our Ackermannization [17] preprocessing pass, which we have extended to support uninterpreted sorts by translating them to bit-vector sorts with sufficiently large bit-widths.

Linear Integer Solver: We have modified our branch-and-bound method to generate ternary clauses instead of binary clauses. The new method is inspired by the Unit-Cube Tests [15]. The ternary clauses are of the form $(x = \bar{a} \vee x \leq \bar{a} - 1 \vee x \geq \bar{a} + 1)$, where x is a variable and $\bar{a}$ is an integer constant that is the closest rounding to the model value given by the linear real arithmetic solver. Moreover, we instruct the DPLL(T) solver to give priority when deciding to the equality literal $(x = \bar{a})$ over the bound literals in the clauses.

Nonlinear Solver: CVC4 uses model-based axioms instantiation techniques (inspired by [16]) for handling nonlinear real and integer arithmetic. We have expanded the axioms set with additional axioms given in [18].

Configurations

This year's version of CVC4 is entering all divisions of the single query, the incremental, the unsat-core, and the model-validation tracks of SMT-COMP 2020. All configurations are compiled with the optional dependencies CLN [2], glpk-cut-log [12] (a fork of GLPK [11]), and CaDiCaL (version 1.2.1). The branch used for all configurations is `smtcomp2020` [4]. For each track, we use a binary that was compiled with different options and the corresponding run script uses different parameters depending on the logic used in the input. For details about the parameters used for each logic, please refer to the run scripts at [5]–[8].

Single Query Track (CVC4): For the Single Query track, we configure CVC4 for optimized reading from non-interactive inputs. We further configure it without proof support. For certain logics, we try different options sequentially (see runscript at [7]).

Incremental Track (CVC4-inc): For the Incremental track, we configured CVC4 for optimized reading from interactive inputs and without proof support. See runscript at [5] for the options used for each logic.

Unsat-Core Track (CVC4-uc): For the Unsat Core track, we configure CVC4 for optimized reading from non-interactive inputs (see runscript at [8]). We further configure it with proof support since the proof infrastructure is used for computing unsat cores.

Model-Validation Track (CVC4-mv): For the model-validation track, we use a similar configuration as for the Single Query track (see runscript at [6]). For `QF_LRA`, we disable the simplification of unconstrained terms since it is not compatible with model generation

Copyright and License

Acknowledgments

CVC4 is supported in part by the organizations listed on our website [3].

References

[1] CaDiCaL. https://github.com/arminbiere/cadical, 2020.
[2] CLN. https://ginac.de/CLN/, 2020.
[3] CVC4 acknowledgments. https://cvc4.github.io/acknowledgements.html, 2020.
[4] CVC4 SMT-COMP 2020 branch. https://github.com/CVC4/CVC4/tree/smtcomp2020, 2020.
[5] CVC4 SMT-COMP 2020 Incremental Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2020/contrib/competitions/smt-comp/run-script-smtcomp-current-incremental, 2020.
[6] CVC4 SMT-COMP 2020 Model Validation Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2020/contrib/competitions/smt-comp/run-script-smtcomp-current-model-validation, 2020.
[7] CVC4 SMT-COMP 2020 Single Query run script. https://github.com/CVC4/CVC4/blob/smtcomp2020/contrib/competitions/smt-comp/run-script-smtcomp-current, 2020.
[8] CVC4 SMT-COMP 2020 Unsat Core Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2020/contrib/competitions/smt-comp/run-script-smtcomp-current-unsat-cores, 2020.
[9] CVC4 source code. https://github.com/CVC4/CVC4, 2020.
[10] CVC4 website. http://cvc4.cs.stanford.edu, 2020.
[11] GLPK. https://www.gnu.org/software/glpk/, 2020.
[12] glpk-cut-log. https://github.com/timothy-king/glpk-cut-log, 2020.
[13] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.
[14] Armin Biere. CaDiCaL at the SAT Race 2019. In *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*, pages 8–9. University of Helsinki, 2019.
[15] Martin Bromberger and Christoph Weidenbach. Fast cube tests for LIA constraint solving. In *IJCAR*, volume 9706 of *Lecture Notes in Computer Science*, pages 116–132. Springer, 2016.
[16] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Trans. Comput. Log.*, 19(3):19:1–19:52, 2018.
[17] Liana Hadarean. *An efficient and trustworthy theory solver for bit-vectors in satisfiability modulo theories*. PhD thesis, Citeseer, 2015.
[18] Ahmed Irfan, Alberto Griggio, Alessandro Cimatti, and Roberto Sebastiani. Mathsat5 (nonlinear) at the smt competition 2019.
[19] Andrew Reynolds, Andres Noetzli, Clark Barrett, and Cesare Tinelli. A decision procedure for string to code point conversion. In *IJCAR*, 2020.
[20] Andrew Reynolds, Maverick Woo, Clark Barrett, David Brumley, Tianyi Liang, and Cesare Tinelli. Scaling up DPLL(T) string solvers using context-dependent simplification. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, pages 453–474, 2017.