CVC4 at the SMT Competition 2019

Clark Barrett[1], Haniel Barbosa[2], Martin Brain[3], Tim King[4], Makai Mann[1], Aina Niemetz[1], Andres Nötzli[1], Alex Ozdemir[1], Mathias Preiner[1], Andrew Reynolds[2], Cesare Tinelli[2], and Yoni Zohar[1]

[1]Stanford University
[2]The University of Iowa
[3]University of Oxford
[4]Google

***Abstract*—This paper is a description of the CVC4 SMT solver as entered into the 2019 SMT Competition. We only list important differences from the 2018 SMT Competition version of CVC4. For further and more detailed information about CVC4, please refer to the original paper [14], 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 / Improvements

The CVC4 configuration entered in the SMT Competition 2019 is an improved and extended version of the version that entered SMT-COMP 2018. Most notably, it features the following extensions.

Eager Bit-Blasting Solver: Last year, we started using CaDiCaL [2, 15] (GitHub master `b44ce4f`) as our SAT back-end for eager bit-blasting. This year, we use CaDiCaL version sr2019 as submitted to the SAT race 2019, which has now been extended to support incremental solving [16]. We updated our bit-vector solver to support CaDiCaL as a back-end for incremental benchmarks. For this year's version, we don't use ABC [1] for bit-blasting since CVC4's bit-blasting solver performs better with the internal bit-blasting infrastructure. We have also optimized our Ackermannization [17] preprocessing pass to generate fewer consistency lemmas.

String Solver: This year's string solver is significantly faster than last year's. Besides tuning the heuristics, we have added more aggressive rewriting and better reductions for extended string operators such as `str.contains` and `str.indexof`, which try to minimize the introduction of new variables. Additionally, CVC4 now reduces certain regular expression to extended string operators.

Floating-Point Solver: CVC4 uses a newer version of SymFPU [13] (commit `8fbe139`) than last year. The updated version primarily fixes a correctness issue.

Configurations

This year's version of CVC4 is entering all divisions of the single query, the incremental, the industry, the unsat-core, and the (experimental) model-validation tracks of SMT-COMP 2019. All configurations are compiled with the optional dependencies CLN [3], glpk-cut-log [12] (a fork of GLPK [11]), CaDiCaL (version sr2019), and CryptoMiniSat version 5.6.3. The commit used for all configurations is tagged with `smtcomp2019` [7]. 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 [4]–[6, 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 [6]).

Incremental Track (CVC4-inc): For the Incremental track, we configured CVC4 for optimized reading from interactive inputs and without proof support. In contrast to last year's version, we use the eager bit-blasting engine with CaDiCaL as a back-end (see runscript at [4]) for the Incremental `QF_BV` track.

Industry Challenge Track (CVC4, CVC4-inc): For the Industry Challenge Track, we use the same configurations as for the Single Query and the Incremental tracks (see runscripts at [4, 6]), depending on whether the division is non-incremental or incremental.

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. Compared to last year, we removed the options for unconstrained simplification [17] for `QF_LRA` and eager bit-blasting for `QF_UFBV` because those options are not currently compatible with unsat cores.

Model-Validation Track (CVC4-mv): For the model-validation track, we use the same configuration as for the Single Query track (see runscript at [5]).

COPYRIGHT

LICENSE

REFERENCES

[1] ABC. https://people.eecs.berkeley.edu/~alanmi/abc/abc.htm, 2019.
[2] CaDiCaL. https://github.com/arminbiere/cadical, 2019.
[3] CLN. https://ginac.de/CLN/, 2019.
[4] CVC4 SMT-COMP 2019 Incremental Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2019/contrib/run-script-smtcomp2019-incremental, 2019.
[5] CVC4 SMT-COMP 2019 Model Validation Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2019/contrib/run-script-smtcomp2019-model-validation, 2019.
[6] CVC4 SMT-COMP 2019 Single Query run script. https://github.com/CVC4/CVC4/blob/smtcomp2019/contrib/run-script-smtcomp2019, 2019.
[7] CVC4 SMT-COMP 2019 tag. https://github.com/CVC4/CVC4/releases/tag/smtcomp2019, 2019.
[8] CVC4 SMT-COMP 2019 Unsat Core Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2019/contrib/run-script-smtcomp2019-unsat-cores, 2019.
[9] CVC4 source code. https://github.com/CVC4/CVC4, 2019.
[10] CVC4 website. http://cvc4.cs.stanford.edu, 2019.
[11] GLPK. https://www.gnu.org/software/glpk/, 2019.
[12] glpk-cut-log. https://github.com/timothy-king/glpk-cut-log, 2019.
[13] SymFPU. https://github.com/martin-cs/symfpu, 2019.
[14] 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.
[15] Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In Tomáš Balyo, Marijn Heule, and Matti Järvisalo, editors, *SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, pages 14–15. University of Helsinki, 2017.
[16] Katalin Fazekas, Armin Biere, and Christoph Scholl. Incremental Inprocessing in SAT Solving. In *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, Lecture Notes in Computer Science, to appear, 2019.
[17] Liana Hadarean. *An efficient and trustworthy theory solver for bit-vectors in satisfiability modulo theories*. PhD thesis, Citeseer, 2015.