CVC4 at the SMT Competition 2018

Clark Barrett[1], Haniel Barbosa[2], Martin Brain[3], Duligur Ibeling[1], Tim King[4], Paul Meng[2], Aina Niemetz[1], Andres Nötzli[1], Mathias Preiner[1], Andrew Reynolds[2], and Cesare Tinelli[2]

[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 2018 SMT Competition. We only list important differences from the 2017 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 2018 is an improved and extended version of the version that entered SMT-COMP 2017. Most notably, it features the following extensions.

Floating-Point Solver: CVC4 now features a floating point solver and thus, for the first time, enters all FP logics of all tracks of the competition. Its FP engine uses SymFPU [13] to translate floating-point operations into bit-vector operations, which are then handed to CVC4's lazy bit-vector engine [17].

Eager Bit-Blasting Solver: Last year, we used CryptoMiniSat [4, 24] version 4 as the back-end SAT solver for CVC4's eager bit-blasting engine. This year, for the first time, CaDiCaL [2, 15] (commit id b44ce4f) serves as our SAT back-end for eager bit-blasting.

Heuristic Approaches for Non-Linear Arithmetic: CVC4 uses techniques for handling non-linear real and integer arithmetic inspired by recent work by Cimatti et al [16]. If a QF_NIA problem cannot easily be solved with that approach, it resorts to turning the input into a bit-vector problem. This year, it uses CaDiCaL as the underlying SAT solver for this approach.

Quantifier Instantiation: For unsatisfiable problems with quantifiers, CVC4 primarily uses conflict-based quantifier instantiation [21] and E-matching. CVC4 additionally implements finite model-finding techniques [22] for satisfiable problems with quantifiers.

Quantified Bit-Vectors: In [19, 20], we present a novel approach for solving quantified bit-vectors based on computing symbolic inverses of bit-vector operators. This approach is now the default for quantified bit-vectors in CVC4.

Strings: This year, CVC4 is entering the non-competitive experimental division QF_SLIA (strings). In this division, CVC4 uses the procedure described in [18] combined with a finite model-finding approach, which searches for strings of bounded length. For handling extended string functions like string contains, substring and replace, CVC4 uses context-dependent simplification techniques as described in [23].

Configurations

This year's version of CVC4 is entering all divisions in the main, application, and unsat core tracks of SMT-COMP 2018. It further enters the non-competitive experimental division QF_SLIA (strings). All configurations are compiled with the optional dependencies ABC [1], CLN [3], glpk-cut-log [12] (a fork of GLPK [11]), CaDiCaL, and CryptoMiniSat version 5. The commit used for all configurations is tagged with `smtcomp2018` [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 certain logics, we try different options sequentially. For details about the parameters used for each logic, please refer to the run scripts.

Main track (CVC4-main): For the main track, we configured CVC4 for optimized reading from non-interactive inputs and without proof support. In contrast to last year's version, we do not use a portfolio configuration for QF_BV since the eager bit-blasting engine with CaDiCaL as a back end in sequential configuration is more efficient. The run script is available at [6].

Application track (CVC4-application): For the application track, we configured CVC4 for optimized reading from interactive inputs and without proof support. The run script is available at [5].

Unsat core track (CVC4-uc): For the unsat core track, we configured CVC4 for optimized reading from non-interactive inputs and with proof support (required for unsat core support). The run script is available at [8].

Experimental (CVC4-experimental-idl-2): Additionally, an experimental configuration, which features a specialized IDL solver, enters the QF_IDL division of the main track. It implements a shortest paths algorithm as an incremental version of the Floyd-Warshall algorithm that can update weights as new edges are added.

COPYRIGHT

LICENSE

REFERENCES

[1] ABC. https://people.eecs.berkeley.edu/~alanmi/abc/abc.htm, 2018.
[2] CaDiCaL. https://github.com/arminbiere/cadical, 2018.
[3] CLN. https://ginac.de/CLN/, 2018.
[4] CryptoMiniSat. https://github.com/msoos/cryptominisat, 2018.
[5] CVC4 SMT-COMP 2018 Application Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2018/contrib/run-script-smtcomp2018-application, 2018.
[6] CVC4 SMT-COMP 2018 Main Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2018/contrib/run-script-smtcomp2018, 2018.
[7] CVC4 SMT-COMP 2018 tag. https://github.com/CVC4/CVC4/releases/tag/smtcomp2018, 2018.
[8] CVC4 SMT-COMP 2018 Unsat Core Track run script. https://github.com/CVC4/CVC4/blob/smtcomp2018/contrib/run-script-smtcomp2018-unsat-cores, 2018.
[9] CVC4 source code. https://github.com/CVC4/CVC4, 2018.
[10] CVC4 website. http://cvc4.cs.stanford.edu, 2018.
[11] GLPK. https://www.gnu.org/software/glpk/, 2018.
[12] glpk-cut-log. https://github.com/timothy-king/glpk-cut-log, 2018.
[13] SymFPU. https://github.com/martin-cs/symfpu, 2018.
[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] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Invariant checking of NRA transition systems via incremental reduction to LRA with EUF. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, pages 58–75, 2017.
[17] Liana Hadarean, Clark Barrett, Dejan Jovanović, Cesare Tinelli, and Kshitij Bansal. A tale of two solvers: Eager and lazy approaches to bit-vectors. In *CAV*, 2014.
[18] Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark Barrett, and Morgan Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 646–662, 2014.
[19] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark Barrett, and Cesare Tinelli. On solving quantified bit-vectors using invertibility conditions. *CoRR*, abs/1804.05025, 2018.
[20] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark Barrett, and Cesare Tinelli. Solving quantified bit-vectors using invertibility conditions. 2018 (to appear).
[21] A. Reynolds, C. Tinelli, and L. de Moura. Finding Conflicting Instances of Quantified Formulas in SMT. In K. Claessen and V. Kuncak, editors, *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design*, pages 195–202, 2014.
[22] A. Reynolds, C. Tinelli, A. Goel, S. Krstic, M. Deters, and C. Barrett. Quantifier Instantiation Techniques for Finite Model Finding in SMT. In M.P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction*, number 7898 in Lecture Notes in Artificial Intelligence, pages 377–391. Springer-Verlag, 2013.
[23] 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.
[24] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.