CVC5 at the SMT Competition 2021

Clark Barrett[1], Haniel Barbosa[3], Martin Brain[4], Gereon Kremer[1], Makai Mann[1], Abdalrhman Mohamed[2], 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 Yoni Zohar[1]

[1]Stanford University
[2]The University of Iowa
[3]Universidade Federal de Minas Gerais

***Abstract*—This paper is a description of the CVC5 SMT solver as entered into the 2021 SMT Competition. CVC5 is the successor of CVC4 [16] and our submission to the SMT Competition serves as a preview of the upcoming release. Here, we briefly discuss the main techniques implemented by CVC5 that are relevant to the competition and how they compare to CVC4. A comprehensive system description is the subject of a future publication. In the meantime, please refer to our website [7] and the source code on GitHub [6] for more information.**

Overview

CVC5 is an 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. CVC5 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).

Features

CVC5 is a CDCL($\mathcal{T}$)-based SMT solver that supports all theories standardized in SMT-LIB. It uses a modified version of MiniSat [27] as its CDCL($\mathcal{T}$) SAT solver. Theory combination is based on the polite combination framework [31, 40] using care graphs [32, 33].

Linear Arithmetic CVC5's solver for linear arithmetic implements a Simplex procedure [26]. It includes heuristics proposed by Griggio [28]. Integers are handled by first solving the real relaxation of the constraints, and then using a combination of cuts from proofs of unsatisfiability [25] and branching to ensure integer solutions [29]. Additionally, the branch-and-bound method can optionally generate lemmas consisting of ternary clauses inspired by unit-cube tests [19].

Non-linear Arithmetic For non-linear arithmetic, we use strategies that are based on the combination of two independent subsolvers. The first subsolver is based on incremental linearization [21], where models are found for the linear abstraction of the input formula, i.e., treating multiplication as an uninterpreted function. Lemma schemas are then used to state properties of multiplication in a counterexample-guided fashion. Details on the lemma schemas used by this subsolver are described in [45]. The second subsolver implements cylindrical algebraic coverings [13] using the polynomial arithmetic and other algebraic routines from libpoly [34].

We primarily invoke incremental linearization for non-linear integer problems, and cylindrical algebraic decomposition for non-linear real problems. We additionally invoke incomplete techniques based on reductions to bit-vectors for non-linear integer problems, and combinations of the two solvers described above for non-linear real arithmetic.

Arrays Like in CVC4, the array solver implements a procedure inspired by the one described in de Moura and Bjørner [23]. Optionally, CVC5 reasons about arrays using an approach proposed by Christ and Hoenicke to lazily instantiate lemmas based on dependencies between arrays that differ in finitely many indices [20].

Bit-Vectors CVC5 features a new bit-blasting bit-vector solver, which allows to use off-the-shelf SAT solvers such as CaDiCaL or CryptoMiniSat [2] as SAT back-ends. In the current version, we use CaDiCaL [17] by default. The new bit-blasting solver seamlessly integrates into the CDCL($\mathcal{T}$) infrastructure of CVC5 and fully supports the combination of bit-vectors with any theory supported by CVC5.

Datatypes For handling quantifier-free constraints over datatypes, we use a rule-based procedure that follows the calculi described in [15, 41]. The procedure incorporates optimizations for sharing selectors over multiple constructors [48].

Floating-Point Arithmetic CVC5 eagerly translates floating-point expressions to the theory of bit-vectors. For that, it integrates SymFPU [18], a C++ library of bit-vector encodings of floating-point operations. SymFPU is also integrated in the SMT solver Bitwuzla [37] and was already used in CVC4. Conversions between real and floating-point numbers are handled lazily.

Strings CVC5's string solver consists of multiple components. At its core, the solver reasons about word equations [35]. The solver supplements reasoning about word equations with reasoning about code points to handle conversions between strings and integers efficiently [50]. The component responsible for extended functions such as string replacement, lazily reduces those functions to word equations after context-dependent simplifications [46]. Skolem variables in the lemmas produced by the reductions reuse existing Skolem variables whenever possible for greater efficiency [51]. The regular expression component unfolds and computes derivatives of regular expressions [36]. The string solver incorporates aggressive simplification rules that rely on abstractions to derive facts about string terms [49]. Finally, the solver detects conflicts eagerly on partial assignments from the SAT solver by computing the congruence-closure and constant prefixes and suffixes of string terms.

Uninterpreted Functions The theory solver for uninterpreted functions resembles Simplify's approach [24] and remains largely unchanged from CVC4. When combined with bit-vectors, CVC5 supports the Ackermannization and eager bit-blasting of constraints involving uninterpreted functions and sorts [30].

Quantifiers For handling logics where quantifiers are present, we rely on heuristic E-matching when they are combined with uninterpreted functions [22]. This technique is supplemented by conflict-based instantiation for detecting when an instantiation is in conflict with the current set of assertions [43]. Our strategy additionally incorporates finite model finding techniques, which are useful for finding satisfiable instances [42]. We additionally rely on enumerative approaches for instantiation when all other techniques are incomplete [47].

For quantifiers over linear arithmetic, we use a specialized counterexample-guided based approach for quantifier instantiation [44]. An extension of this technique is used for quantified bit-vector logics [38]. For other quantified logics in pure background theories, e.g., over floating-point or non-linear arithmetic, we use new techniques for syntax-guided quantifier instantiation [39].

Decision Heuristic In addition to MiniSat's decision heuristic, CVC5 implements a separate heuristic that uses the original Boolean structure of the input to keep track of the *justified* parts of the input constraints, i.e., the parts where it can infer the value of terms based on a (partial) assignment to subterms. For decisions, it traverses assertions that are not satisfied under the current assignment, computing the desired values (starting with true as the desired value for the root) for each term until it finds a literal that has not been assigned and would contribute towards a desired value. The heuristic optionally prioritizes assertions that led to decisions that resulted in a conflict. This heuristic is a reimplementation and extension of a heuristic implemented in CVC4 [14].

Unsat Cores CVC5 implements two approaches to compute unsatisfiable cores: (i) assumption-based unsat cores (ii) proof-based unsat cores. Both approaches use the new proof infrastructure featured in CVC5. CVC5's proof infrastructure allows it to generate fine-grained proofs for unsatisfiable problems. The assumption-based approach uses MiniSat's support for computing unsatisfiable assumptions. CVC5 uses the proof infrastructure to track the preprocessing of assertions, sends the constraints as assumptions to MiniSat, and retrieves the list of unsatisfiable assumptions after running its regular solving procedure. The proof-based approach uses the proof infrastructure to track preprocessing and the reasoning done by the SAT solver. After the main solving procedure finishes, it extracts the unsat core from the proof.

CONFIGURATIONS

CVC5 is entering all divisions of the single query, the incremental, the unsat-core, and the model-validation tracks of SMT-COMP 2021.

The branch used for all configurations is `smtcomp2021` [11]. For each track, we use a binary that was compiled specifically for the track 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 in the competition branch [8, 9, 10, 12]. All configurations are compiled with the optional dependencies CLN [1], glpk-cut-log [4] (a fork of GLPK [5]), CaDiCaL (commit `88623ef`), SymFPU (commit `8fbe139`), and libpoly (commit `6309f7a`).

Single Query Track (CVC5) For the Single Query track, we configure CVC5 for optimized reading from non-interactive inputs. For certain logics, we try different options sequentially (see runscript at [10]).

Incremental Track (CVC5-inc) For the Incremental track, we configured CVC5 for optimized reading from interactive inputs and use the default options for most logics. See the runscript [8] for more details.

Unsat-Core Track (CVC5-uc) For the Unsat Core track, we configure CVC5 for optimized reading from non-interactive inputs and use options similar to the ones used for the Single Query Track (see runscript [12] for details). The submission uses assumption-based unsat cores.

Model-Validation Track (CVC5-mv) For the model-validation track, we use a similar configuration as for the Single Query track (see runscript [9] for details). For `QF_LRA`, we disable the simplification of unconstrained terms since it is not compatible with model generation.

COPYRIGHT AND LICENSE

ACKNOWLEDGMENTS

CVC5 is supported in part by the organizations listed on our website [3].

References

[1] CLN. https://ginac.de/CLN/, 2020.

[2] CryptoMiniSat. https://github.com/msoos/cryptominisat, 2020.

[3] cvc5 acknowledgments. https://cvc5.github.io/acknowledgements.html, 2020.

[4] glpk-cut-log. https://github.com/timothy-king/glpk-cut-log, 2020.

[5] GLPK. https://www.gnu.org/software/glpk/, 2020.

[6] CVC5 source code. https://github.com/cvc5/cvc5, 2021.

[7] CVC5 website. https://cvc5.github.io/, 2021.

[8] cvc5 SMT-COMP 2021 Incremental Track run script. https://github.com/cvc5/cvc5/blob/smtcomp2021/contrib/competitions/smt-comp/run-script-smtcomp-current-incremental, 2021.

[9] cvc5 SMT-COMP 2021 Model Validation Track run script. https://github.com/cvc5/cvc5/blob/smtcomp2021/contrib/competitions/smt-comp/run-script-smtcomp-current-model-validation, 2021.

[10] cvc5 SMT-COMP 2021 Single Query run script. https://github.com/cvc5/cvc5/blob/smtcomp2021/contrib/competitions/smt-comp/run-script-smtcomp-current, 2021.

[11] cvc5 SMT-COMP 2021 branch. https://github.com/cvc5/cvc5/tree/smtcomp2021, 2021.

[12] cvc5 SMT-COMP 2021 Unsat Core Track run script. https://github.com/cvc5/cvc5/blob/smtcomp2021/contrib/competitions/smt-comp/run-script-smtcomp-current-unsat-cores, 2021.

[13] Erika Ábrahám, James H. Davenport, Matthew England, and Gereon Kremer. Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. *J. Log. Algebraic Methods Program.*, 119:100633, 2021. doi: 10.1016/j.jlamp.2020.100633. URL https://doi.org/10.1016/j.jlamp.2020.100633.

[14] Kshitij Bansal. A branching heuristic in cvc4 smt solver. https://kshitij.io/articles/cvc4-branching-heuristic.pdf, 2012.

[15] Clark Barrett, Igor Shikanian, and Cesare Tinelli. An abstract decision procedure for a theory of inductive data types. *JSAT*, 3(1-2):21–46, 2007. URL http://jsat.ewi.tudelft.nl/content/volume3/JSAT3_3_Barrett.pdf.

[16] 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.

[17] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[18] Martin Brain, Florian Schanda, and Youcheng Sun. Building better bit-blasting for floating-point problems. In *TACAS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *LNCS*, pages 79–98. Springer, 2019. doi: 10.1007/978-3-030-17462-0_5. URL https://doi.org/10.1007/978-3-030-17462-0_5.

[19] 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.

[20] Jürgen Christ and Jochen Hoenicke. Weakly equivalent arrays. In *FroCos*, volume 9322 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2015.

[21] 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 Axel Legay and Tiziana Margaria, editors, *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*, volume 10205 of *Lecture Notes in Computer Science*, pages 58–75, 2017. doi: 10.1007/978-3-662-54577-5_4. URL https://doi.org/10.1007/978-3-662-54577-5_4.

[22] Leonardo Mendonça de Moura and Nikolaj Bjørner. Efficient e-matching for SMT solvers. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2007. doi: 10.1007/978-3-540-73595-3_13. URL https://doi.org/10.1007/978-3-540-73595-3_13.

[23] Leonardo Mendonça de Moura and Nikolaj Bjørner. Generalized, efficient array decision procedures. In *FMCAD*, pages 45–52. IEEE, 2009.

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

[25] Isil Dillig, Thomas Dillig, and Alex Aiken. Cuts from proofs: a complete and practical technique for solving linear inequalities over integers. *Formal Methods Syst. Des.*, 39(3):246–260, 2011.

[26] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[27] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[28] Alberto Griggio. *An Effective SMT Engine for Formal Verification*. PhD thesis, University of Trento, Italy, 2009.

[29] Alberto Griggio. A practical approach to satisfiability modulo linear integer arithmetic. *Journal on Satisfiability,*

Boolean Modeling and Computation, 8(1-2):1–27, 2012.

[30] Liana Hadarean. *An efficient and trustworthy theory solver for bit-vectors in satisfiability modulo theories*. PhD thesis, Citeseer, 2015.

[31] Dejan Jovanovic and Clark W. Barrett. Polite theories revisited. In *LPAR (Yogyakarta)*, volume 6397 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2010.

[32] Dejan Jovanovic and Clark W. Barrett. Sharing is caring: Combination of theories. In *FroCoS*, volume 6989 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2011.

[33] Dejan Jovanovic and Clark W. Barrett. Being careful about theory combination. *Formal Methods Syst. Des.*, 42(1):67–90, 2013.

[34] Dejan Jovanovic and Bruno Dutertre. Libpoly: A library for reasoning about polynomials. In *Proc. 15th International Workshop on Satisfiability Modulo Theories (SMT 2017)*, number 1889, 2017.

[35] Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark W. Barrett, and Morgan Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 646–662. Springer, 2014.

[36] Tianyi Liang, Nestan Tsiskaridze, Andrew Reynolds, Cesare Tinelli, and Clark W. Barrett. A decision procedure for regular membership and length constraints over unbounded strings. In *FroCos*, volume 9322 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2015.

[37] Aina Niemetz and Mathias Preiner. Bitwuzla at the SMT-COMP 2020. *CoRR*, abs/2006.01621, 2020. URL https://arxiv.org/abs/2006.01621.

[38] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Solving quantified bit-vectors using invertibility conditions. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 236–255. Springer, 2018. doi: 10.1007/978-3-319-96142-2_16. URL https://doi.org/10.1007/978-3-319-96142-2_16.

[39] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Syntax-guided quantifier instantiation. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 145–163. Springer, 2021. doi: 10.1007/978-3-030-72013-1_8. URL https://doi.org/10.1007/978-3-030-72013-1_8.

[40] Silvio Ranise, Christophe Ringeissen, and Calogero G. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In *FroCoS*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64. Springer, 2005.

[41] Andrew Reynolds and Jasmin Christian Blanchette. A decision procedure for (co)datatypes in SMT solvers. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 197–213. Springer, 2015. doi: 10.1007/978-3-319-21401-6_13. URL https://doi.org/10.1007/978-3-319-21401-6_13.

[42] Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstic, Morgan Deters, and Clark W. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2013. doi: 10.1007/978-3-642-38574-2_26. URL https://doi.org/10.1007/978-3-642-38574-2_26.

[43] Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura. Finding conflicting instances of quantified formulas in SMT. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, pages 195–202. IEEE, 2014. doi: 10.1109/FMCAD.2014.6987613. URL https://doi.org/10.1109/FMCAD.2014.6987613.

[44] Andrew Reynolds, Tim King, and Viktor Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods Syst. Des.*, 51(3):500–532, 2017. doi: 10.1007/s10703-017-0290-y. URL https://doi.org/10.1007/s10703-017-0290-y.

[45] Andrew Reynolds, Cesare Tinelli, Dejan Jovanovic, and Clark W. Barrett. Designing theory solvers with extensions. In Clare Dixon and Marcelo Finger, editors, *Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings*, volume 10483 of *Lecture Notes in Computer Science*, pages 22–40. Springer, 2017. doi: 10.1007/978-3-319-66167-4_2. URL https://doi.org/10.1007/978-3-319-66167-4_2.

[46] Andrew Reynolds, Maverick Woo, Clark W. Barrett, David Brumley, Tianyi Liang, and Cesare Tinelli. Scaling up DPLL(T) string solvers using context-dependent simplification. In *CAV (2)*, volume 10427 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2017.

[47] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software,*

ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II, volume 10806 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 2018. doi: 10.1007/978-3-319-89963-3_7. URL https://doi.org/10.1007/978-3-319-89963-3_7.

[48] Andrew Reynolds, Arjun Viswanathan, Haniel Barbosa, Cesare Tinelli, and Clark W. Barrett. Datatypes with shared selectors. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 591–608. Springer, 2018. doi: 10.1007/978-3-319-94205-6_39. URL https://doi.org/10.1007/978-3-319-94205-6_39.

[49] Andrew Reynolds, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. High-level abstractions for simplifying extended string constraints in SMT. In *CAV (2)*, volume 11562 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2019.

[50] Andrew Reynolds, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. A decision procedure for string to code point conversion. In *IJCAR (1)*, volume 12166 of *Lecture Notes in Computer Science*, pages 218–237. Springer, 2020.

[51] Andrew Reynolds, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. Reductions for strings and regular expressions revisited. In *FMCAD*, pages 225–235. IEEE, 2020.