cvc5 at the SMT Competition 2021

Clark Barrett¹, Haniel Barbosa³, Martin Brain⁴, Gereon Kremer¹, Makai Mann¹, Abdalrhman Mohamed², Mudathir Mohamed², Aina Niemetz¹, Andres Nötzli¹, Alex Ozdemir¹, Mathias Preiner¹, Andrew Reynolds², Ying Sheng¹, Cesare Tinelli², and Yoni Zohar¹

¹Stanford University  
²The University of Iowa  
³Universidade 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.

FEATURES

cvc5 is a CDCL(T)-based SMT solver that supports all theories standardized in SMT-LIB. It uses a modified version of MiniSat [27] as its CDCL(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 Björner [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(T) infrastructure of cvc5 and fully supports the combination of bit-vectors with any theory supported by cvc5.

Data types 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 [55]. 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 [43]. 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 [11], glpk-cut-log [4] (a fork of GLPK [5]), CaDiCaL (commit 88623ef), SymFPU (commit 8fbee139), and libploy (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 configure 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**

CVC5 is copyright 2021 by its authors and contributors and their institutional affiliations. For a full list of authors, refer to the AUTHORS and THANKS files distributed with the source code [6].

The source code of CVC5 is open and available to students, researchers, software companies, and everyone else to study, to modify, and to redistribute original or modified versions; distribution is under the terms of the modified BSD license. Please note that CVC5 can be configured (however, by default it is not) to link against some GPLed libraries, and therefore the use of these builds may be restricted in non-GPL-compatible projects. For more information about CVC5’s license refer to the actual license text as distributed with its source code [6].

**Acknowledgments**

CVC5 is supported in part by the organizations listed on our website [8].
REFERENCES

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

