cvc5 at the SMT Competition 2022

Haniel Barbosa[3], Clark Barrett[1], Martin Brain[4], Gereon Kremer[1], Hanna Lachnitt[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[5]

[1]Stanford University
[2]The University of Iowa
[3]Universidade Federal de Minas Gerais
[4]City, University of London
[5]Bar-Ilan University

***Abstract*—This paper is a description of the cvc5 SMT solver as entered into the 2022 SMT Competition. Here, we briefly summarize the main techniques implemented by cvc5 that are relevant. For more comprehensive information please refer to the tool paper about cvc5 [15], our website [7], and the source code on GitHub [6].**

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. For unsatisfiable formulas, cvc5 can be used to produce proofs in multiple formats [16]. cvc5 is an open and extensible SMT engine, and it can be used as a stand-alone tool or as a library. There is 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 [28] as its CDCL($\mathcal{T}$) SAT solver. Theory combination is based on the polite combination framework [33, 41] using care graphs [34, 35].

Linear Arithmetic cvc5's solver for linear arithmetic implements a Simplex procedure [27]. It includes heuristics proposed by Griggio [29]. Integers are handled by first solving the real relaxation of the constraints, and then using a combination of cuts from proofs of unsatisfiability [26] and branching to ensure integer solutions [30]. Additionally, the branch-and-bound method can optionally generate lemmas consisting of ternary clauses inspired by unit-cube tests [20].

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 [22], 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 [46]. The second subsolver implements cylindrical algebraic coverings [13] using the polynomial arithmetic and other algebraic routines from libpoly [36].

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 The array solver implements a procedure inspired by the one described in de Moura and Bjørner [24]. 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 [21].

Bit-Vectors cvc5's bit-vector solver uses bit-blasting and supports off-the-shelf SAT solvers such as CaDiCaL or CryptoMiniSat [2] as SAT back-ends. In the current version, we use CaDiCaL [18] 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 [17, 42]. The procedure incorporates optimizations for sharing selectors over multiple constructors [49].

Floating-Point Arithmetic cvc5 eagerly translates floating-point expressions to the theory of bit-vectors. For that, it integrates SymFPU [19], a C++ library of bit-vector encodings of floating-point operations. 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 [37]. The solver supplements reasoning about word equations with reasoning about code points to handle conversions between strings and integers efficiently [51]. The component responsible for extended functions such as string replacement, lazily reduces those functions to word equations after context-dependent simplifications [47]. Skolem variables in the lemmas produced by the reductions reuse existing Skolem variables whenever possible for greater efficiency [52]. The regular expression component unfolds and computes derivatives of regular expressions [38]. The string solver incorporates aggressive simplification rules that rely on abstractions to derive facts about string terms [50]. 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 [25] and remains largely unchanged. When combined with bit-vectors, cvc5 supports the Ackermannization and eager bit-blasting of constraints involving uninterpreted functions and sorts [31].

Quantifiers For handling logics where quantifiers are present, we rely on heuristic E-matching when they are combined with uninterpreted functions [23]. This technique is supplemented by conflict-based instantiation for detecting when an instantiation is in conflict with the current set of assertions [44]. Our strategy additionally incorporates finite model finding techniques, which are useful for finding satisfiable instances [43]. We additionally rely on enumerative approaches for instantiation when all other techniques are incomplete [48].

For quantifiers over linear arithmetic, we use a specialized counterexample-guided based approach for quantifier instantiation [45]. An extension of this technique is used for quantified bit-vector logics [39]. 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 [40].

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 cvc5's proof infrastructure. cvc5's proof infrastructure generates 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 2022. We also enter two variants to the proof exhibition track as explained below.

The branch used for all configurations is `smtcomp2022` [11]. We use a binary optimized for reading input from files in a competition setting for all tracks but the incremental track. For the incremental track, we use a binary optimized for reading from interactive inputs. For each track, we use a dedicated run script, which calls cvc5 with parameters that depend 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 `8bc2c3b`), SymFPU (commit `8fbe139`), and libpoly (commit `1383809`).

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) 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) 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.

Proof Exhibition Track (cvc5 and cvc5-lfsc) For the proof exhibition track, cvc5 uses a new flexible proof-producing architecture [16]. cvc5 can check proofs internally or produce proofs in a format that can be checked using an external tool. The internal proof checker checks proofs during construction. If a user does not want to trust cvc5's internal checker and desires to check the proofs with an external checker, they can use the other formats that cvc5 supports, namely LFSC (described below) and Alethe.[1] The internal proofs can also be printed in a DOT format to facilitate visualization.[2]

cvc5 participates with two entries:

[1] See https://cvc5.github.io/docs/cvc5-1.0.0/proofs/output_alethe.html

[2] See https://cvc5.github.io/docs/cvc5-1.0.0/proofs/output_dot.html. One can use a default DOT visualizer or the dedicated proof visualizer available at https://ufmg-smite.github.io/proof-visualizer/

- the default CVC5 entry produces proofs in the *internal proof format* of CVC5.
  The format contains proof rules that very closely represent how CVC5 is solving the problem.[3] A proof rule application has the form

  (RULENAME F1 ... Fn :args t1 ... tm)

  where F1 ... Fn are the *children* proof rule applications, t1 ... tm are the *arguments* terms, and RULENAME is a function from the children and arguments to a *conclusion* term. A proof is directed acyclic graph of proof rule applications. Seen as a tree, the root necessarily has the negation of the input formula as the conclusion, proven to be unsatisfiable.
  No external checker is used. The proofs are rather checked during construction in CVC5 itself, so that if an invalid proof is produced, CVC5 fails with an error message.
- the CVC5-lfsc entry produces proofs using the LFSC framework [53]. LFSC is a logical framework, based on Edinburgh LF [32], which was explicitly designed to facilitate the production and checking of fine-grained proofs in SMT. It comes with a small and efficient proof checker,[4] which is generic in the sense that it takes as input both a proof term p and a *proof signature*, a definition of the data types and proof rules used to construct p. The checker verifies that p is well-formed with respect to the provided signature. We have defined proof signatures for most of the theories supported by CVC5.[5] These definitions can be combined together as needed to define a proof system for any combination of those theories. When emitting proofs in LFSC, CVC5 includes all the relevant signatures as a preamble to the proof term.

Copyright and License

Acknowledgments

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

[3]The rules are described in detail at https://cvc5.github.io/docs/cvc5-1.0.0/proofs/proof_rules.html.

[4]Available at https://github.com/cvc5/LFSC.

[5]Available at https://github.com/cvc5/cvc5/tree/main/proofs/lfsc/signatures.

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 2022 Incremental Track run script. https://github.com/cvc5/cvc5/blob/smtcomp2022/contrib/competitions/smt-comp/run-script-smtcomp-current-incremental, 2022.

[9] cvc5 SMT-COMP 2022 Model Validation Track run script. https://github.com/cvc5/cvc5/blob/smtcomp2022/contrib/competitions/smt-comp/run-script-smtcomp-current-model-validation, 2022.

[10] cvc5 SMT-COMP 2022 Single Query run script. https://github.com/cvc5/cvc5/blob/smtcomp2022/contrib/competitions/smt-comp/run-script-smtcomp-current, 2022.

[11] cvc5 SMT-COMP 2022 branch. https://github.com/cvc5/cvc5/tree/smtcomp2022, 2022.

[12] cvc5 SMT-COMP 2022 Unsat Core Track run script. https://github.com/cvc5/cvc5/blob/smtcomp2022/contrib/competitions/smt-comp/run-script-smtcomp-current-unsat-cores, 2022.

[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] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS (1)*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.

[16] Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, and Clark Barrett. Flexible proof production in an industrial-strength SMT solver. In *IJCAR*, 2022.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

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

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

[26] 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.

[27] 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.

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

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

[30] Alberto Griggio. A practical approach to satisfiability modulo linear integer arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1-2):1–27, 2012.

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

[32] Robert Harper, Furio Honsell, and Gordon D. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, 1993.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Syntax-guided quantifier instantiation. In Jan Friso Groote and Kim Guld-

strand 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

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

[53] Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean, and Cesare Tinelli. SMT proof checking using a logical framework. *Formal Methods in System Design*, 42(1):91–118, 2013. doi: 10.1007/s10703-012-0163-3.