CVC4

Kshitij Bansal[1], Clark Barrett[1], François Bobot[2], Christopher L. Conway[1], Morgan Deters[1], Liana Hadarean[1], Dejan Jovanović[1], Tim King[1], Andrew Reynolds[2], and Cesare Tinelli[2]

[1]New York University [2]University of Iowa

Abstract. CVC4 is the latest version of the Cooperating Validity Checker. A joint project of NYU and U Iowa, CVC4 aims to support the useful feature set of CVC3 and SMT-LIBv2 while optimizing the design of the core system architecture and decision procedures to take advantage of recent engineering and algorithmic advances. CVC4 represents a completely new code base; it is a from-scratch rewrite of CVC3, and many subsystems have been completely redesigned. We describe the system architecture, subsystems of note, and discuss some applications and continuing work.

1 Introduction

The Cooperating Validity Checker series has a long history, starting with the Stanford Validity Checker (SVC) [3], and continuing with the Cooperating Validity Checker (CVC) [25], CVC Lite [1], and CVC3 [5].

CVC4 is the new version, the fifth generation of this validity checker line that is now celebrating fifteen years of heritage. It represents a complete re-evaluation of the core architecture to be both performant and to serve as a cutting-edge research vehicle for the next several years. Rather than taking CVC3 and redesigning problem parts, we've taken a clean-room approach, starting from scratch. Before using any designs from CVC3, we have thoroughly scrutinized, vetted, and updated them. Many parts of CVC4 bear only a superficial resemblance, if any, to their correspondent in CVC3. However, CVC4 is fundamentally similar to CVC3 and many other modern SMT solvers: it is a DPLL(T) solver [18], with a SAT solver at its core and a delegation path to different decision procedure implementations, each in charge of solving formulas in some background theory.

2 Design of CVC4

CVC4 is organized around a central core of *engines:*

This work partially supported by the NSF (CCF–0644299, CCF–0914956, CNS–1049495, and 0914877), AFOSR (FA9550–09–1–0596 and FA9550–09–1–0517), SRC 2008–TJ–1850, and MIT Lincoln Laboratory. This system description draws from and builds on the CAV 2011 CVC4 tool paper [2].

- The *SMT Engine* serves as the main outside interface point to the solver. Known in previous versions of CVC as the *ValidityChecker*, the *SMT Engine* has public functions to push and pop solving contexts, manipulate a set of currently active assumptions, and check the validity of a formula, as well as functions to request proofs and generate models. This engine is responsible for setting up and maintaining all user-related state.
- The *Prop Engine* manages the propositional solver at the core of CVC4. This, in principle, allows different SAT solvers to be plugged into CVC4. (At present, only MiniSat is supported, due to the fact that a SAT solver must be modified to dispatch properly to SMT routines.)
- The *Decision Engine* provides support for alternative decision heuristics. CVC4 supports both an "internal" heuristic, which simply uses the SAT solver's internal decision heuristic, as well as custom heuristics which are coded within the *Decision Engine*.
- The *Theory Engine* serves as an "owner" of all decision procedure implementations, giving a single point of interface between the rest of CVC4 and decision procedure implementations. As is common in the research field, these implementations are referred to as *theories* and all are derived from the base class *Theory* and implement a common interface that the *Theory Engine* uses.
- The *Equality Engine* is a reusable class that manages information about equalities and disequalities. It has a number of nice features including high performance, the ability to perform congruence closure on selected operators, and a variety of call-back mechanisms for informing clients when certain terms become equal or disequal.

CVC4 uses *Theory* objects similar to those suggested by modern DPLL(T) literature [18] and used in other solvers. CVC4's *Theory* class is responsible for checking consistency of the current set of assertions, and propagating new facts based on the current set of assertions.

CVC4 incorporates numerous *managers* in charge of managing subsystems:

- The *Node Manager* is one of the busiest parts of CVC4, in charge of the creation and deletion of all expressions ("nodes") in the prover. *Node* objects are immutable and unique; the creation of an already-extant *Node* results in a reference to the original. Node data is reference-counted (the *Node* class itself is just a reference-counted smart pointer to node data) and subject to reclamation by the *Node Manager* when no longer referenced; for performance reasons, this is done lazily.
- The *Shared Term Manager* is in charge of all shared terms in the system. Shared terms are detected by the *Theory Engine* and registered with this manager, and this manager broadcasts new equalities between shared terms.
- The *Context Memory Manager* is in charge of maintaining a coherent, backtrackable data context for the prover. At its core, it is simply a region memory manager, from which new memory regions can be requested ("pushed") and destroyed ("popped") in LIFO order. These regions contain saved state for a number of heap-allocated objects, and when a pop is requested, these heap

objects are "restored" from their backups in the region. This leads to a nice, general mechanism to do backtracking without lots of *ad hoc* implementations in each theory; this is highly useful for rapid prototyping. However, as a general mechanism, it must be used sparingly; as it is often more efficient to perform specialized backtracking manually within a theory using a lighter-weight method.

2.1 Preprocessing

CVC4 includes a sophisticated preprocessor with several passes:

- Nonclausal simplification looks at the structure of the formula to be checked and identifies literals that must be true based on the Boolean structure. These literals are then processed: when possible a true equation is solved for a variable and this variable is replaced everywhere in the formula; otherwise, if an equation has a constant value on one side, then constant propagation is done, replacing the other side with the constant value wherever it appears.
- ITE simplification looks for opportunities to simplify the formula by analyzing if-then-else terms. It uses a restricted form of the algorithm described in [23] as well as the simplification algorithm described in [12].
- Unconstrained value simplification looks for *unconstrained* values: variables which appear only once in the formula. Often, it is possible to replace the immediate predecessor of an unconstrained value with a new unconstrained value. The idea is described in [7, 9].
- Theory preprocessing applies theory-specific preprocessing rewrites that have the potential to simplify the formula based on the literals identified in the nonclausal simplification phase.

2.2 Theories

CVC4 incorporates newly-designed and implemented decision procedures for its theories:

- The theory of uninterpreted functions is little more than a wrapper around an instance of the *Equality Engine*. It does congruence closure and theory propagation on literals from its theory.
- The theory of arithmetic handles both linear real and linear integer arithmetic. It uses a highly-tuned simplex implementation based on [17] as well as a variety of integer techniques adapted from [20].
- The theory of arrays is based on lazy lemma instantiation as described in [15, 22].
- The theory of bit-vectors uses a new approach which combines lazy bit-blasting with in-processing using an algebraic solver. Currently, the algebraic solver is no more than an *Equality Engine*, but we hope to improve it significantly going forward.
- The theory of datatypes implements a decision procedure for mutually recursive inductive datatypes based on [4].

– A "theory" of quantifiers handles skolemization and instantiation. It is based on current best-practices for quantifier instantiation [19, 13] and also incorporates some new features based on counter-example detection and finite model finding. A related module allows quantified axioms to be used as rewrite rules. For certain theories that can be specified using rewrite rules, the performance is far superior than if they were simply specified as quantified axioms.

Theory combination relies on polite combination [21] and care functions [22]. Care functions are computed by the uninterpreted function theory and the array theory.

2.3 Proofs

CVC4's proof system is designed to support LFSC proofs [24], and is also designed to have *absolutely zero footprint* in memory and time when proofs are turned off at compile-time.

2.4 Library API

As CVC4 is meant to be used via a library API, there's a clear division between the public, outward-facing interface, and the private, inward-facing one. This is a distinction that wasn't as clear in the previous version; installations of CVC3 required the installation of *all* CVC3 header files, because public headers depended on private ones to function properly. Not so in CVC4, where only a subset of headers declaring public interfaces are installed on a user's machine. Inward-facing interfaces are not even exported by the library.

Further, we have decided "to take our own medicine." Our own tools, *including CVC4's parser and main command-line tool,* link against the CVC4 library in the same way that any end-user application would (and therefore does not have access to internal, undocumented interfaces). This helps us ensure that the library API is complete—since if it is not, the command-line CVC4 tool is missing functionality, too, an omission we catch quickly.

2.5 Theory modularity

Theory objects are designed in CVC4 to be highly *modular:* they do not employ global state, nor do they make any other assumptions that would inhibit their functioning as a client to another decision procedure. In this way, one *Theory* can instantiate and send subqueries to a completely subservient client *Theory* without interfering with the main solver flow.

2.6 Support for concurrency

CVC4's infrastructure has been designed to make the transition to multiprocessor and multicore hardware easy, and we currently have an experimental lemma-sharing portfolio version of CVC4. We intend CVC4 to be a good vehicle for other

research ideas in this area as well. In part, the modularity of theories (above) is geared toward this—the absence of global state and the immutability of expression objects clearly makes it easier to parallelize operations. Similarly, the *Theory* API specifically includes the notion of *interruptibility*, so that an expensive operation (*e.g.*, theory propagation) can be interrupted if work in another thread makes it irrelevant. Current work being performed at NYU and U Iowa is investigating different ways to parallelize SMT; the CVC4 architecture provides a good experimental platform for this research, as it does not need to be completely re-engineered to test different concurrent solving strategies.

3 Conclusion

SMT solvers are currently an area of considerable research interest. Barcelogic [6], CVC3 [5], MathSat [10] OpenSMT [11], Yices [16], and Z3 [14] are examples of modern, currently-maintained, popular SMT solvers. OpenSMT and CVC3 are open-source, and CVC3, Yices, and Z3 are the only ones to support all of the defined SMT-LIB logics, including quantifiers.

CVC4 aims to follow in CVC3's footsteps as an open-source theorem prover supporting this wide array of background theories. CVC3 supports all of the background theories defined by the SMT-LIB initiative, and provides proofs and counterexamples upon request; CVC4 aims for full compliance with the new SMT-LIB version 2 command language and backward compatibility with the CVC presentation language.

In this way, we hope CVC4 can largely be a drop-in replacement for CVC3, with a cleaner and more consistent library API, a more modular, flexible core, a far smaller memory footprint, and better performance characteristics.

CVC4's superior performance (compared to CVC3) is apparent by comparing their performance in SMT-COMP 2012 [8]. CVC4 outperformed CVC3 in every category. CVC4 performs competitively with other solvers and was the winner in the QF_UFLRA division.

Our goal with CVC4 is to provide a high-performing, fully-featured SMT solver. CVC4 now supports most of the features that CVC3 supported as well as some new ones (the rewrite-rule system for example). We expect to continue to improve the existing features while actively developing new ones. New features under development include support for nonlinear arithmetic, a theory of strings, and a theory for reasoning about floating point values.

References

1. Clark Barrett and Sergey Berezin. CVC Lite: A new implementation of the Cooperating Validity Checker. In Rajeev Alur and Doron A. Peled, editors, *Proceedings of the* 16^{th} *International Conference on Computer Aided Verification (CAV '04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 515–518. Springer-Verlag, July 2004. Boston, Massachusetts.

2. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the* 23^{rd} *International Conference on Computer Aided Verification (CAV '11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, July 2011. Snowbird, Utah.
3. Clark Barrett, David Dill, and Jeremy Levitt. Validity checking for combinations of theories with equality. pages 187–201. Springer-Verlag, 1996.
4. Clark Barrett, Igor Shikanian, and Cesare Tinelli. An abstract decision procedure for a theory of inductive data types. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:21–46, 2007.
5. Clark Barrett and Cesare Tinelli. CVC3. In Werner Damm and Holger Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany.
6. Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodrguez-Carbonell, and Albert Rubio. The Barcelogic SMT solver. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 294–298. Springer Berlin / Heidelberg, 2008.
7. R. Brummayer. *Efficient SMT Solving for Bit-Vectors and the Extensional Theory of Arrays*. PhD thesis, Johannes Kepler University, November 2009.
8. R. Bruttomesso, D. Cok, and A. Griggio. SMT-COMP 2012: the 2012 edition of the satisfiability modulo theories competition. `http://smtcomp.sourceforge.net/2012/`.
9. Roberto Bruttomesso. *RTL Verification: From SAT to SMT(BV)*. PhD thesis, University of Trento, 2008. Available at `http://www.inf.unisi.ch/postdoc/bruttomesso`.
10. Roberto Bruttomesso, Alessandro Cimatti, Anders Franzn, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT solver. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 299–303. Springer Berlin / Heidelberg, 2008.
11. Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT solver. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153. Springer Berlin / Heidelberg, 2010.
12. Jerry R. Burch. Techniques for verifying superscalar microprocessors. In *Proceedings of the 33rd annual Design Automation Conference*, DAC '96, pages 552–557, New York, NY, USA, 1996. ACM.
13. Leonardo de Moura and Nikolaj Bjørner. Efficient E-Matching for SMT solvers automated deduction CADE-21. In Frank Pfenning, editor, *Automated Deduction CADE-21*, volume 4603 of *Lecture Notes in Computer Science*, chapter 13, pages 183–198. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007.
14. Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.
15. Leonardo de Moura and Nikolaj Bjørner. Generalized, efficient array decision procedures. In *Formal Methods in Computer-Aided Design, 2009*, pages 45–52. IEEE, November 2009.
16. Bruno Dutertre and Leonardo de Moura. The YICES SMT solver. `http://yices.csl.sri.com/tool-paper.pdf`.

17. Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proceedings of the 18th Computer-Aided Verification conference*, volume 4144 of *LNCS*, pages 81–94. Springer-Verlag, 2006.
18. Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): fast decision procedures. pages 175–188. Springer, 2004.
19. Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Annals of Mathematics and Artificial Intelligence*, 55(1-2):101–122, February 2009.
20. A. Griggio. *An Effective SMT Engine for Formal Verification*. PhD thesis, DISI, University of Trento, November 2009.
21. Dejan Jovanović and Clark Barrett. Polite theories revisited. In Christian G. Fermüller and Andrei Voronkov, editors, *Proceedings of the* 17^{th} *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '10)*, volume 6397 of *Lecture Notes in Computer Science*, pages 402–416. Springer, October 2010. Yogyakarta, Indonesia.
22. Dejan Jovanović and Clark Barrett. Being careful about theory combination. *Formal Methods in System Design*, pages 1–24, 2012.
23. Hyondeuk Kim, Fabio Somenzi, and HoonSang Jin. Efficient Term-ITE conversion for satisfiability modulo theories theory and applications of satisfiability testing - SAT 2009. volume 5584 of *Lecture Notes in Computer Science*, chapter 20, pages 195–208. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2009.
24. Duckki Oe, Andrew Reynolds, and Aaron Stump. Fast and flexible proof checking for SMT. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, SMT '09, pages 6–13, New York, NY, USA, 2009. ACM.
25. Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: A cooperating validity checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the* 14^{th} *International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer-Verlag, July 2002. Copenhagen, Denmark.