CVC3

Clark Barrett[1] and Cesare Tinelli[2]

[1]New York University, barrett@cs.nyu.edu
[2]University of Iowa, tinelli@cs.uiowa.edu

Abstract. CVC3, a joint project of NYU and U Iowa, is the new and latest version of the Cooperating Validity Checker. CVC3 extends and builds on the functionality of its predecessors and includes many new features such as support for additional theories, an abstract architecture for Boolean reasoning, and SMT-LIB compliance. We describe the system and discuss some applications and continuing work.

1 Introduction

Like its predecessors, SVC [5], CVC [12], and CVC Lite [1], CVC3 is an automatic theorem prover for the *Satisfiability Modulo Theories* (SMT) problem: given an input formula ϕ in first order logic, CVC3 attempts to determine the validity (or dually, the satisfiability) of ϕ with respect to one or more background theories.

CVC3 builds on the architecture and features of the successful CVC and CVC Lite systems, but it also differs in important ways. First of all, the project is under new management: it is being developed at NYU and the University of Iowa (unlike its predecessors, all of which were hosted at Stanford University). Second, the system is mature enough now that it seemed best to drop the "Lite" moniker. It is called CVC3 because it is the third major release of a system with the CVC name. Most importantly, many new features have been added and most of the code has been revised or rewritten. For these reasons, a new major release and accompanying system description seemed appropriate.

2 System Description

A high-level view of CVC3's architecture is shown in Fig. 1. CVC3 provides several different user interfaces including high-level API's for both C and C++, an interactive command-driven interface, and a file interface. The Main API supports two main types of operations: formula creation and methods for validity/satisfiability checking. The main deduction engine is called the Search Engine. Its role is to link the Boolean reasoning capabilities of the DPLL engine with the theory reasoning capabilities of the Theory Solver. The DPLL engine relies on a Boolean SAT solver to do its work and the Theory Solver relies on a set of decision procedures, one for each supported theory.

Because of space limitations, we focus on new features in this paper, referring the reader to previous work for a discussion of the more basic features of the system. The new features can be broadly partitioned into three categories: the search engine, new theories, and enhanced usability.

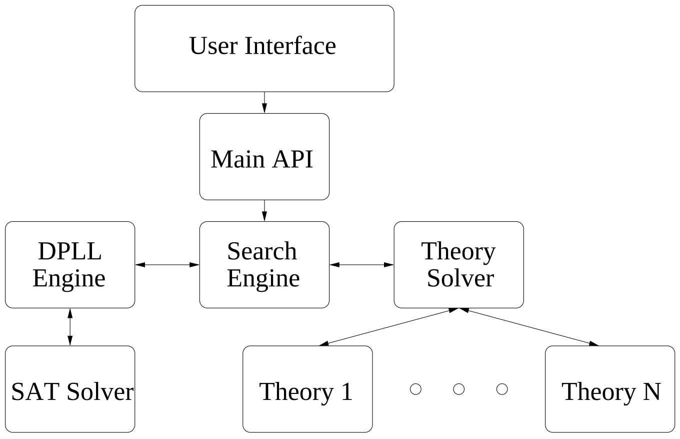

Fig. 1. CVC3 System design

2.1 The Search Engine

CVC3 features a new Search Engine. The Search Engine processes incoming queries, first using standard techniques to convert them into equisatisfiable formulas in conjunctive normal form. The Boolean structure is then fed to the DPLL Engine. One of the primary reasons for developing a new Search Engine was to make it easy to plug in different implementations of the DPLL Engine. To do this, a simple abstract API was developed based on the Extended Abstract DPLL Modulo Theories framework [3]. Ideas developed in this theoretical framework, such as theory propagation and splitting on demand, made it possible to implement a simple API that was rich enough to be practical and efficient. Implementations of this API are largely shielded from the rest of the system and communicate with the Search Engine using a simple minimal interface which references only basic data-structures for Boolean variables, literals, and clauses.

One measure of success is that we were quickly able to integrate two different SAT solvers. In fact, our experience has been that the main difficulty is not in implementing the API, but in adapting the SAT solvers to support necessary features like dynamic addition of clauses and variables. The SAT solvers currently available in CVC3 are zchaff [11] and MiniSat [7]. Another measure of success is that with the new Search Engine, CVC3 outperforms CVC Lite on nearly all benchmarks, typically by a factor of 2 or 3, but by up to an order of magnitude on benchmarks with significant Boolean structure.

2.2 New Theories

Abstract Data Types. While its predecessors could reason about simple aggregate data types like records and tuples, CVC3 has the ability to reason about arbitrary recursive and mutually recursive data types. A simple example of a recursive data type is the *list* type from LISP with constructors *null* and *cons* and selectors *car* and *cdr*. A simple example of a query that CVC3 can solve over this type is: $\forall x : \mathit{list}.\, x = \mathit{null} \vee \exists yz.\, x = \mathit{cons}(y,z)$. The implementation is based on our abstract decision procedure described in [4].

Bitvectors. Support for a theory of bitvectors was a late addition to the CVC Lite system. In CVC3, the bitvector theory has been largely reworked with a resulting substantial improvement in performance. However, the implementation is still rather naive and based on a simple combination of pre-processing and bit-blasting. We consider improving the efficiency of bitvector reasoning to be an important research challenge.

Quantifiers. CVC3 treats quantified formulas as if they belonged to a separate "quantifier" theory. This convenient mechanism allows CVC3 to use a special strategy for quantified formulas: existential formulas are skolemized and then passed back to the main theory solver for additional processing; universal formulas are accumulated and a set of heuristics is used to instantiate the formulas with ground terms from other literals known to the theory solver. CVC3 contains a new instantiation mechanism that extends the "matching" techniques of the Simplify theorem prover [6]. CVC3 is significantly better than CVC Lite on formulas with quantifiers and our experiments on the SMT-LIB benchmarks indicate that it can solve more problems than other instantiation-based systems [9].

2.3 Enhanced Usability

SMT-LIB. In order to support the SMT-LIB initiative, a powerful translation module was added to CVC Lite. It has been updated and improved in CVC3. This module is capable of translating benchmarks to and from the SMT-LIB format. The most difficult part of this is inferring the correct SMT-LIB logic based on syntactic properties of the benchmark. CVC3 has been used to verify the correct logic categorization of all benchmarks currently in the SMT-LIB library, and it is currently the standard for checking the syntax and categorization of new benchmarks submitted to the library.

Model Generation. An important feature of CVC3 is that it can produce concrete models after a satisfiable query. For example, instead of reporting $x \neq y$, CVC3 can assign actual values to x and y, such as $x = 0$ and $y = 1$. This is useful for tools that use CVC3 as a back-end and need to provide meaningful feedback to the user. It should be mentioned that this feature was already present in CVC Lite, but it was added after the system description was published and so is worth emphasizing here.

Incremental Use. It has always been possible to use the CVC tools incrementally using a stack-based push and pop mechanism. Several new features have been added to aid incremental use. First of all, the Minisat implementation of the DPLL Engine has been enhanced to be incremental. This means that it is possible to reuse lemmas learned from one query in another related query. Second, it is possible to search for additional models after a model has been found by using a "continue" command. Finally, it is also possible to search for models that satisfy an additional assumption. This is implemented with a command called "restart". The restart command is is useful for refining abstractions and has been implemented in such a way that the work done in finding the first model can be reused, which is important for efficiency.

3 Conclusion

CVC3 aims to continue the tradition of its predecessors by providing a free, robust, automatic, and feature-rich tool suitable for a variety of research and industrial applications. Some applications of previous versions of CVC include a proof-producing decision procedure for HOL Light [10]; a verification tool for C programs [8], a translation validator for optimizing compilers [2], and a study on the verification of clock synchronization algorithms [13].

We expect that these and similar future applications will find CVC3 even more useful. In particular, we currently have collaborations in place with research groups at the University of Dublin, Microsoft Research, and Rockwell-Collins on using CVC3 respectively within an extended static checker for Java, an automated unit test generator for .NET programs, and a model checker for programs written in the dataflow language Lustre.

There is still much that we plan to do to improve CVC3. Current work includes improvements to the arithmetic, bitvector, and quantifier theories. New theories under consideration include a theory of strings, a theory of sets, and a theory of subtypes. One important enhancement we expect to make soon is to allow user-defined symbols to have polymorphic types. We also plan to improve the current support for proofs and models. Finally, of course, we would like to continue to improve overall performance of the system.

CVC3 has an active user and development community. More information, including instructions for downloading and installing the tool, can be found on the CVC3 web site at `http://www.cs.nyu.edu/acsys/cvc3`.

References

1. C. Barrett and S. Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *CAV*, pages 515–518, 2004.
2. C. Barrett, Y. Fang, B. Goldberg, Y. Hu, A. Pnueli, and L. Zuck. TVOC: A translation validator for optimizing compilers. In *CAV*, pages 291–295, 2005.
3. C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on demand in SAT modulo theories. In *LPAR*, 2006.
4. C. Barrett, I. Shikanian, and C. Tinelli. An abstract decision procedure for satisfiability in the theory of recursive data types. In *PDPAR*, 2006.
5. C. W. Barrett, D. L. Dill, and J. R. Levitt. Validity checking for combinations of theories with equality. In *FMCAD*, pages 187–201, 1996.
6. D. L. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem-prover for program checking. Technical Report HPL-2003-148, HP System Research Center, 2003.
7. N. Eén and N. Sörensson. An extensible sat-solver. In *SAT*, 2003.
8. J.-C. Filliâtre and C. Marché. Multi-Prover Verification of C Programs. In *ICFEM*, pages 15–29, 2004.
9. Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification condisions using satisfiability modulo theories. In *CADE*, July 2007. To appear.
10. S. McLaughlin, C. Barrett, and Y. Ge. Cooperating theorem provers: A case study combining HOL-Light and CVC Lite. In *PDPAR*, pages 43–51, 2006.
11. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *DAC*, 2001.
12. A. Stump, C. W. Barrett, and D. L. Dill. CVC: A cooperating validity checker. In *CAV*, pages 500–504, 2002.
13. A. Tiu, D. Barsotti, and L. Prensa. Verification of clock synchronization algorithms: Experiments on a combination of deductive tools. In *AVOCS*, 2005.