CVC Lite: A New Implementation of the Cooperating Validity Checker*

Category B

Clark Barrett[1] and Sergey Berezin[2]

[1]New York University, barrett@cs.nyu.edu
[2]Stanford University, berezin@stanford.edu

Abstract. We describe a tool called CVC Lite (CVCL), an automated theorem prover for formulas in a union of first-order theories. CVCL supports a set of theories which are useful in verification, including uninterpreted functions, arrays, records and tuples, and linear arithmetic. New features in CVCL (beyond those provided in similar previous systems) include a library API, more support for producing proofs, some heuristics for reasoning about quantifiers, and support for symbolic simulation primitives.

1 Introduction

Decision procedures for decidable fragments of first-order logic continue to attract users and interest in a wide variety of verification efforts.

CVC Lite (CVCL) is a tool for determining the validity (or satisfiability) of first-order formulas over a union of specific useful theories. It replaces the original Cooperating Validity Checker (CVC) [7], which, in turn, was a successor to the Stanford Validity Checker (SVC) [4]. The name does not imply that the new system is less powerful than CVC, but rather was chosen because after learning from our experience with CVC, we felt we could create a tool which, without sacrificing functionality, would be smaller, faster, and easier to use and maintain.

Although CVCL is a work in progress, in many respects it has already validated our vision and rewarded the effort involved in a reimplementation. In particular, the code base is one third the size of CVC, the performance is comparable, and it has been used and enhanced by a number of people outside the core group of developers. In addition, CVCL has many new features, not found in any of the previous systems.

In this paper, we will describe the theory and features of CVCL, with an emphasis on what is new as compared to the previous systems (especially CVC). We begin with a brief overview of the system and the theories which are currently supported in CVCL. Then we describe the features which are new in CVCL and conclude with some example applications.

* This research was supported by a grant from Intel Corporation and by National Science Foundation CCR-0121403.

2 Overview

CVCL accepts as input one or more assertion formulas and a query formula. It then checks whether the assertion formulas imply the query formula. Each formula must be a first-order formula whose parameters (non-logical symbols) must be from among the theories listed in the next section.

The algorithm used depends on the Nelson-Oppen method for combining decision procedures [6] and the implementation is based closely on an algorithm whose correctness is verified in the first author's Ph.D. thesis [3].

Although there is limited support for quantifiers in CVCL (see below), the algorithm is complete only for quantifier-free formulas. As with its predecessor, CVCL uses advanced SAT-based search heuristics and has the ability to produce a proof when a formula is successfully validated.

3 The Theories of CVCL

3.1 Equality with Uninterpreted Functions

The simplest supported theory is one which contains an arbitrary number of functions and predicates which are "uninterpreted", meaning that the theory does not provide any information about them other than that they are functions and predicates. Because the set of non-logical symbols in this theory varies according to the formulas being checked, the user must specify the set of such functions and predicates for a particular run of CVCL.

3.2 Arrays

CVCL includes a theory of abstract arrays with two operations, *read* and *write* which can be used to read from a location in an abstract array or to create a new array by *writing* a new value to a location in an existing array.

3.3 Records and Tuples

CVCL formulas can include simple aggregate datatypes like records and tuples. These are handled with a simple decision procedure for a set of operations used to create, read from, and write to these datatypes (much like the array operations).

3.4 Arithmetic

As with its predecessors, CVCL can decide the theory of linear arithmetic over the reals. However, CVCL also has some additional capabilities. The first is the ability to deal with linear arithmetic over integers. In fact, CVCL can reason about linear expressions over any combination of real and integer variables.

The other extension implemented in CVCL is the ability to handle some nonlinear arithmetic. Nonlinear expressions are transformed into a normal form, making it possible to verify simple identities like $(a+b)(a-b) = a^2 - b^2$. However, the nonlinear capabilities of CVCL are still very limited.

3.5 Additional Theories

Currently, new decision procedures are being developed for inductive datatypes, a subset of set theory, and a theory of bit-vectors.

4 New Features

4.1 Library API

One of the main features lacking in both SVC and CVC was a library interface. Interaction with the old systems was done using a small custom command language. Commands were either typed in manually or provided through a scripting mechanism.

CVCL has the same command language interface, but we also designed an abstract interface into CVCL from the start. The methods in this API mimic the command language, so that it is easy to move from one mode of interaction to the other. In fact, the command language interface is implemented using the API, minimizing the chance that the two modes of interaction will behave differently.

The API is available both as an abstract C++ class and as a set of C functions. It has been successfully used as a library from C++, and the C interface has been successfully used by the foreign function interface of other languages including Prolog and Ocaml.

4.2 Proof Support for Efficient Boolean Reasoning

A major feature of the original CVC system was the ability to produce a proof artifact as the result of successfully validating a formula. However, CVC could only produce proofs when using a slow search heuristic. When using advanced SAT-based heuristics, which are essential on large formulas, CVC was unable to produce a proof because it depended on an external SAT solver and had no way to extract a proof from this solver.

CVCL overcomes this difficulty by integrating a custom SAT solver and including proof rules for the kinds of reasoning done in modern efficient Boolean SAT solvers [2]. This enables CVCL to use advanced techniques like clause learning and conflict-directed backtracking while still producing proofs.

4.3 Quantifiers

One of the most significant new features of CVCL is native support for quantifiers. Adding quantifiers necessarily makes the logic undecidable, but in many practical examples, even very simple heuristics for quantifier instantiation can be sufficient.

The current heuristic used by CVCL is to collect the set of terms that have occurred in some previous formula, and then use these terms to instantiate the quantified variables of similar type. This is a very close reimplementation of the heuristic used by Das and Dill [5] for solving quantified formulas arising in predicate abstraction.

4.4 Symbolic Simulation

A primitive interface for symbolic simulation was built into CVC, and successfully applied to applications in hardware verification [1]. CVCL provides a more extensive and intuitive interface to symbolic simulation primitives.

5 Conclusion

Since becoming available in August 2003, CVCL has been downloaded by many research groups and used in a wide variety of verification efforts in both hardware and software.

One representative example is the work on compiler validation being done at NYU. CVCL is used to verify the verification conditions generated by a tool which checks the correctness of transformations done by an optimizing compiler [8].

CVCL has an active user and development community. More information, including instructions for downloading and installing the tool, is available at the CVCL web page: http://verify.stanford.edu/CVCL.

References

1. Husam Abu-Haimed, Sergey Berezin, and David L. Dill. Strengthening invariants by symbolic consistency testing. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*. Springer, 2003.
2. Clark Barrett and Sergey Berezin. A Proof-Producing Boolean Search Engine. In *CADE-19 Workshop: Pragmatics of Decision Procedures in Automated Reasoning (PDPAR)*, July 2003. Miami, Florida, USA.
3. Clark W. Barrett. *Checking Validity of Quantifier-Free Formulas in Combinations of First-Order Theories*. PhD thesis, Stanford University, 2003.
4. Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. Validity Checking for Combinations of Theories with Equality. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods In Computer-Aided Design (FMCAD)*, volume 1166 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, November 1996. Palo Alto, California.
5. Satyaki Das and David L. Dill. Counter-example based predicate discovery in predicate abstraction. In *Formal Methods in Computer-Aided Design*. Springer-Verlag, November 2002.
6. Greg Nelson and Derek Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–57, 1979.
7. Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: A Cooperating Validity Checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer-Verlag, 2002. Copenhagen, Denmark.
8. Lenore Zuck, Amir Pnueli, Benjaming Goldberg, Clark Barrett, Yi Fang, and Ying Hu. Translation and run-time validation of optimized code. *(to appear in) Formal Methods in Systems Design*, 2004. Preliminary version in *Third Workshop on Runtime Verification (RV), 2002.*