CVC: a Cooperating Validity Checker

Aaron Stump, Clark W. Barrett, and David L. Dill

Computer Systems Laboratory, Stanford University, Stanford, CA 94305, USA
E-mail: {stump,barrett,dill}@cs.stanford.edu
Phone: +1 650 725 3646, Fax: +1 650 725 6949

Abstract. Decision procedures for decidable logics and logical theories have proven to be useful tools in verification. This paper describes the CVC ("Cooperating Validity Checker") decision procedure. CVC implements a framework for combining subsidiary decision procedures for certain logical theories into a decision procedure for the theories' union. Subsidiary decision procedures for theories of arrays, inductive datatypes, and linear real arithmetic are currently implemented. Other notable features of CVC are the incorporation of the high-performance Chaff solver for propositional reasoning, and the ability to produce independently checkable proofs for valid formulas.

1 Introduction

Decision procedures for decidable logics and logical theories have been used successfully in several approaches to verification. They play an important role in verification based on interactive theorem provers (e.g., PVS [8]), where decidable subgoals that arise in proofs of system correctness can be automatically discharged by decision procedures, thus reducing the burden on the user. They have also been used in more automatic approaches to verification, where verification problems are reduced to validity checking problems, typically involving very large formulas (e.g., [9]).

CVC is a high-performance system for checking validity of formulas in a relatively rich decidable logic. Atomic formulas are applications of predicate symbols like $<$ and $=$ to first-order terms like $\mathtt{x} + 2 * \mathtt{y}$ and $\mathtt{car}(\mathtt{cons}(\mathtt{x}, \mathtt{L}))$. Formulas are then the usual boolean combinations (built using `AND`, `OR`, `NOT`, etc.) of atomic formulas. CVC's language provides predicate and function symbols which are convenient for modelling systems like hardware, protocols, and software. CVC is implemented in around 150K lines of C++.

CVC is the successor to the Stanford Validity Checker (SVC) [1]. In addition to the ability to produce proofs and the incorporation of an efficient SAT solver, CVC has many improvements over SVC. The codebase is much more robust and extensible. The C++ Standard Template Library (STL) is used for efficient data structures. Such seemingly minor features as the syntax for the input language and the quality of the error messages have been greatly improved, resulting in a much more usable system. The following is an example of CVC input:

```
list : TYPE = DATATYPE cons (car : REAL, cdr : list), null END;
L1, L2 : list;
x, y : REAL;
P : [ [REAL, REAL]-> BOOLEAN];

QUERY (x = 2 * y - 1) AND (L1 = L2 WITH car := x) =>
      P(x + y, car(L1)) => P(3 * y - 1, x);
```

The example first declares an inductive datatype of lists. Then it declares some uninterpreted constants and an uninterpreted binary predicate `P`. It then queries a formula, which in this case is valid. The `WITH` operator performs functional updating of a data structure.

2 Cooperating decision procedures

Early work by Nelson and Oppen showed that under certain restrictions, independent decision procedures for quantifier-free logical theories in classical first-order logic with equality can be combined to obtain a decision procedure for the union of the theories [7]. The most basic restriction is that the theories may not share function and predicate symbols other than the equality symbol. The union of the theories can contain terms like `car(L) + 3 * x` which have function symbols from the signatures of more than one theory. A variant of the Nelson-Oppen approach is implemented in CVC [2]. The subsidiary decision procedures currently implemented are for the following theories.

Arrays: The theory of arrays implemented [10] has function symbols for reading from a location i in an array a (syntax: `a[i]`) and functionally updating an array a to contain a given value v at a given index i (syntax: `a WITH [i] := v`). Arrays are extensional, which leads to validity of non-trivial equalities between updated arrays such as (assuming `a` is an array)

```
((a WITH [1] := 100) WITH [2] := 200)   =
((a WITH [2] := 200) WITH [1] := 100).
```

Inductive datatypes: CVC allows the user to declare inductive datatypes like lists and trees. Inductive datatypes are determined by a set of constructors, like *cons* and *null*, which construct members of the datatype out of some constituent elements (possibly none at all); and selectors, like *car* and *cdr*, which retrieve constituent elements from members of the datatype. Selectors are considered partial functions, so `car(null)` is considered to be undefined. When a datatype is declared, testers like `cons?` and `null?` are automatically added. `c?(x)` is true iff `x` was constructed using constructor `c`. CVC's language has special syntax for tuples and records, which are special cases of inductive datatypes.

Linear real arithmetic: The theory of linear real arithmetic has the usual function symbols for addition, subtraction, and arithmetic negation, as well as for multiplication and division by a constant. There are also the usual predicate symbols for arithmetic comparison. CVC implements a version of Fourier-Motzkin variable elimination to handle inequalities.

3 Proofs

CVC can optionally produce proofs for every formula it reports valid. The proofs are represented using a variant of the Edinburgh Logical Framework (LF) [5], extended with features for more conveniently representing multi-arity functions like the tuple-forming operator and n-ary addition [12]. The proofs can be efficiently checked by a proof checker called **flea** [11], which ships with CVC.

4 Chaff

Given the great advances that have been made in propositional SAT solving tools in the last decade, much greater performance on problems with boolean structure can be achieved by incorporating a modern SAT solver. CVC incorporates the Chaff SAT solver [6] to do its propositional reasoning. The Chaff code is modified to assert CVC literals (atomic formulas or their negations) in its search for a satisfying assignment. When the rest of CVC discovers a contradiction, a conflict clause is added to Chaff containing the relevant assertions. CVC determines which assertions are relevant to the contradiction by reusing the infrastructure that produces proofs in order to track assumptions [3]. This approach greatly improves performance.

5 Performance

Figure 5 compares CVC and its predecessor SVC on benchmarks from processor verification. Size is the size in kilobytes of the formula represented with maximal sharing of common subexpressions in ASCII. Running times are in seconds on an 850MHz PIII. CVC is faster than SVC on all but a handful of the examples. All but the last three examples were part of SVC's suite of benchmarks, and hence are among the examples that SVC could be expected to perform best on.

test	size (Kb)	SVC time	CVC time
fb_12_11	10	1.0	0.2
fb_5_12	11	4.2	0.3
fb_6_12	8	1.1	0.2
dlx-dmem	71	0.2	1.8
dlx-pc	87	0.2	0.9
dlx-regfile	71	0.2	3.8
pp-bloaddata-a	32	0.6	1.6
pp-bloaddata	31	8.8	4.1
pp-dmem2	30	8.6	1.4
pp-invariant	29	0.3	0.2
ibm-full-5	350	16.1	2.3
ibm-full-10	370	15.0	2.3
bool_dlx2_aa	238	$>$ 10000	0.7

6 Related work

CVC is similar to the ICS system [4]. ICS implements a version of Shostak's algorithm for combining decision procedures, which is less general than the framework implemented in CVC. Other features of CVC that distinguish it from ICS are

- incorporation of a state-of-the-art SAT solver
- the ability to produce independently verifiable proofs
- support for arbitrary inductive datatypes
- implementation in C++ (ICS is written in Ocaml)

Other cooperating decision procedures include:

- Simplify at Compaq SRC (http://research.compaq.com/SRC/esc/Simplify.html)
- STeP at Stanford (http://www-step.stanford.edu/)
- Vampyre at Berkeley (http://www-cad.eecs.berkeley.edu/ rupak/Vampyre/)

7 Final remarks

A Linux executable together with basic examples and documentation is freely available at `http://verify.stanford.edu/CVC`. We thank the anonymous reviewers for their comments. This work was supported under ARPA/Air Force contract F33615-00-C-1693 and NSF grants CCR-9806889 and CCR-0121403.

References

1. C. Barrett, D. Dill, and J. Levitt. Validity checking for combinations of theories with equality. In M. Srivas and A. Camilleri, editors, *Formal Methods In Computer-Aided Design*, volume 1166 of *LNCS*, pages 187–201. Springer-Verlag, 1996.
2. C. Barrett, D. Dill, and A. Stump. A Framework for Cooperating Decision Procedures. In David McAllester, editor, *17th International Conference on Computer Aided Deduction*, volume 1831 of *LNAI*, pages 79–97. Springer-Verlag, 2000.
3. C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In *14th International Conference on Computer-Aided Verification*, 2002.
4. J. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: integrated canonizer and solver. In G. Berry, H. Comon, and A. Finkel, editors, *13th International Conference on Computer-Aided Verification*, 2001.
5. R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.
6. M. Moskewicz, C. Madigan, Y. Zhaod, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *39th Design Automation Conference*, 2001.
7. G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–57, 1979.
8. S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *LNAI*, pages 748–752. Springer-Verlag, 1992.
9. J. Skakkebæk, R. Jones, and D. Dill. Formal verification of out-of-order execution using incremental flushing. In *10th International Conference on Computer Aided Verification*, 1998.

10. A. Stump, C. Barrett, D. Dill, and J. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *16th IEEE Symposium on Logic in Computer Science*, pages 29–37. IEEE Computer Society, 2001.
11. A. Stump and D. Dill. Faster Proof Checking in the Edinburgh Logical Framework. In *18th International Conference on Automated Deduction*, 2002.
12. A. Stump and D. Dill. Producing Proofs from an Arithmetic Decision Procedure in Elliptical LF. In *3rd International Workhsop on Logical Frameworks and Meta-Languages*, 2002. (acceptance pending).