Cooperating Theorem Provers: A Case Study Combining HOL-Light and CVC Lite

Sean McLaughlin [a] Clark Barrett [b] Yeting Ge [b]

[a] *Department of Computer Science, Carnegie Mellon University,*
seanmcl@cmu.edu

[b] *Department of Computer Science, New York University,*
{barrett,yeting}@cs.nyu.edu

Abstract

This paper is a case study in combining theorem provers. We define a derived rule in HOL-Light, `CVC_PROVE`, which calls CVC Lite and translates the resulting proof object back to HOL-Light. As a result, we obtain a highly trusted proof-checker for CVC Lite, while also fundamentally expanding the capabilities of HOL-Light.

Key words: HOL, CVC, theorem proving, proof checking

1 Introduction and History

A significant barrier to progress in theorem proving is the fact that it is generally difficult to share results among different theorem provers. One attempt to address this issue is the nascent Logosphere project [1]. This promises to be a database of theorems in varying formats with a systematic translation mechanism between the various logics [8]. Another example is the Ωmega system [2] that can import proofs of some other systems (eg. TPS) and uses other theorem provers for proof search in certain domains (eg. Otter [3] for first order logic).

Another difficulty arises from the fact that different theorem provers put different emphases on proofs and correctness. Using an untrusted external prover as an oracle might compromise soundness which is unacceptable in many cases. Harrison and Théry[11] describe a so-called "skeptic" approach in which external proofs from Maple are translated and reproduced within HOL.

In the hope of encouraging further progress along these lines, this paper describes a case study using the "skeptic" approach in which proof terms generated by the automatic theorem prover CVC Lite are translated into corresponding proofs in the interactive theorem prover HOL-Light. Though

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the general approach is not new, this particular coupling of theorem provers presents unique challenges and benefits.

HOL-Light is a general-purpose theorem prover with a very small trusted core. In contrast, CVC Lite takes a more pragmatic approach, resulting in a much larger trusted core. By linking the two provers, HOL-Light is able to take advantage of CVC Lite's speed and decision procedures. At the same time, by using HOL-Light as a proof-checker, our confidence in results obtained by CVC Lite can be increased significantly.

After a brief description of each system, we explain the implementation and what was required to translate proofs from CVC Lite to HOL-Light. We then give some results including a class of problems that were solvable in HOL-Light using `CVC_PROVE` but were unsolvable otherwise. Other less successful examples are given for comparison. We also show how an array theory can be easily added to HOL-Light with the help of CVC Lite. Finally, we discuss potential applications and future work.

1.1 HOL-Light

HOL-Light [9] is an interactive theorem prover descended from the LCF projects [7,13] and the HOL4 theorem prover [4]. All the theorems are created by a core set of 10 primitive inference rules such as modus ponens and reflexivity. These inference rules are realized as functions in the OCaml language. Other rules of inference (called *derived rules*) are conservatively derived from these primitive rules. They are conservative in the sense that they consist only of increasingly sophisticated applications of the primitive rules, and thus do not expand the power of the system. Using OCaml, the user may program new rules (e.g., decision procedures) without compromising the soundness of the system. The core consists of just about 1500 lines of OCaml. HOL-Light has been used extensively by its author to verify hardware designs at Intel [10]. A large body of mathematics has been formalized in the system, from the construction of the real numbers to basic results in transfinite set theory and real and complex analysis.

1.2 CVC Lite

CVC Lite [5] is an automatic proof-producing theorem prover for decidable first order theories. It is derived from the SVC and CVC projects at Stanford University [6,14]. The logical core differs in many ways from the HOL-Light kernel. For example, as speed is a design goal of the system, there are many more primitive inference rules in CVC Lite. In fact, there are over one hundred rules alone for the theory of real linear arithmetic. (Contrast this number with the 10 total inference rules of HOL-Light, where the reals are constructed from the axiom of infinity.) The trusted code base is correspondingly larger, over 3000 lines being used to solve problems of linear real arithmetic.

2 Implementation

Proofs are represented in CVC Lite as tree-like data structures [1]. Nodes are labeled with the name of an inference rule and the arguments to that rule. These arguments can be types, terms, or other proof objects. We implement a HOL-Light derived rule for each CVC Lite inference rule and translate the proof tree depth first, calling the corresponding HOL-Light rule as each CVC Lite rule is encountered. (An example follows.) Thus, a bug in CVC Lite would not compromise the HOL-Light system. A false proof generated by CVC Lite would simply fail to translate into HOL-Light.

2.1 Translating Types and Terms

Given that the CVC Lite logic is close to a subset of the HOL-Light logic, translating types and terms is straightforward. Each system has a notion of Boolean and real types. Terms in CVC are first order, a sub-language of HOL-Light's higher order object language. Thus translating terms is easy as well.

2.2 Translating Proofs

A proof in CVC Lite is a *proof term* in the sense of the Curry-Howard Isomorphism [12]. The proof itself corresponds directly to a simple functional program. Thus, the translation process can be seen as executing this "program" in HOL-Light.

As an illustration, below is a proof of the proposition $\neg(p \rightarrow q) \leftrightarrow (p \wedge \neg q)$ in CVC Lite's proof format:

```
(iff_trans(NOT (p => q), NOT (NOT p OR q), (p AND NOT q),
           basic_subst_op(NOT (p => q), NOT (NOT p OR q),
                          rewrite_implies(p, q)),
           rewrite_not_or(NOT (NOT p OR q))))
```

CVC Lite's proof format is a lisp-like representation of a derivation tree with the outermost expression representing the root of the tree. Each node of the tree is an application of an inference rule, and its children are either parameters or premises to the inference rule. This proof uses the following four inference rules [2]

$$(1)\quad \frac{}{\vdash (\alpha \rightarrow \beta) \leftrightarrow (\neg\alpha \vee \beta)}\quad \texttt{rewrite\_implies}(\alpha,\ \beta)$$

[1] The only complication being variable binding.

[2] As usual, the expression below the bar represents the conclusion, and the expressions above the bar represent premises.

$$(2)\quad \frac{\vdash \alpha \leftrightarrow \beta}{\vdash \phi[\alpha] \leftrightarrow \phi[\beta]}\quad \texttt{basic\_subst\_op}(\alpha,\ \beta)$$

$$(3)\quad \frac{}{\vdash \neg(\alpha \vee \beta) \leftrightarrow (\neg\alpha \wedge \beta)}\quad \texttt{rewrite\_not\_or}(\neg(\alpha \vee \beta))$$

$$(4)\quad \frac{\vdash \alpha \leftrightarrow \beta \quad \vdash \beta \leftrightarrow \gamma}{\vdash \alpha \leftrightarrow \gamma}\quad \texttt{iff\_trans}(\alpha,\beta,\gamma)$$

Notice that each inference rule has one or more parameters. In the representation of the derivation tree produced by CVC Lite, the values for these parameters at a particular node are taken from the first few children of the node. The rest of the node's children are derivation trees for the premises to the inference rule (if any). The corresponding proof tree looks like this (the parameters to the inference rules are omitted for clarity):

$$\dfrac{\dfrac{\dfrac{}{\vdash (p \to q) \leftrightarrow (\neg p \vee q)}\ (1)}{\vdash \neg(p \to q) \leftrightarrow \neg(\neg p \vee q)}\ (2) \qquad \dfrac{}{\vdash \neg(\neg p \vee q) \leftrightarrow (p \wedge \neg q)}\ (3)}{\vdash \neg(p \to q) \leftrightarrow (p \wedge \neg q)}\ (4)$$

In order to translate these proof terms to HOL-Light, we have a HOL-Light derived rule for each CVC Lite rule encountered in the proof tree. Thus, *translation* corresponds to function application. We have a rule for every inference rule in CVC Lite. We then combine these rules in a recursive procedure that translates the proofs in a depth first traversal of the proof tree. If there are no errors, the translation of the root proof node yields the desired HOL-Light theorem.

3 Results

We focused on two classes of problems solvable by both HOL-Light and CVC Lite: Boolean satisfiability formulas and real arithmetic formulas.

3.1 Boolean Satisfiability

To evaluate the effectiveness of using `CVC_PROVE` for Boolean satisfiability in HOL-Light, we looked at the well-known pigeonhole problems: given $n-1$ sets of n pigeon-holes, arranged in n rows of $n-1$ columns and given that no column can contain more than one pigeon, find a contradiction to the assertion that each row can contain a pigeon. For $n = 3$, the corresponding Boolean

satisfiability formula is:

$$(\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_5) \wedge$$
$$(\neg x_3 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_4) \wedge$$
$$(\neg x_2 \vee \neg x_6) \wedge (\neg x_4 \vee \neg x_6) \wedge$$
$$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6).$$

This is a notoriously difficult class of problems for typical Boolean satisfiability methods. The following table[3] gives times for CVC Lite running alone (but still producing proofs), HOL-Light running alone, and HOL-Light using CVC Lite and performing the translation.

n	CVC Lite	HOL-Light	`CVC_PROVE`
2	0.10	4.5	1.75
3	0.18	13	10
4	0.90	34	43
5	2.9	*	210
6	19	*	980
7	238	*	4308

The empty entries under "HOL Light" are intractable in that system. Even the example with $n = 5$ ran for over 4 hours before we killed the process. We thus expand the power of HOL-Light using the external system CVC Lite.

3.2 Real Arithmetic

The first problems we investigated with the translation process were terms of real linear arithmetic. The HOL-Light decision procedure `REAL_ARITH` was, at the time, very slow. Using `CVC_PROVE` on such problems tended to produce good speed improvement, typically 2 to 3 times as fast as the unaided `REAL_ARITH`. John Harrison, the author of HOL-Light, later optimized `REAL_ARITH`. With the new arithmetic procedure, `CVC_PROVE` is around six times slower. This was a case where optimizing the original decision procedure was more effective than using an external tool.

[3] All times are in seconds, running on a 1GH Pentium III running FreeBSD 5.2

4 The Theory of Arrays

The experiments documented above arise from theories that exist in both theorem provers. A more interesting application of translation is to theories for which decision procedures do not yet exist in one of the provers. For instance, CVC Lite has a well developed theory of arrays. This theory does not exist in the current HOL-Light version. As an alternative to implementing a decision procedure for arrays in HOL-Light we extended the current translation mechanism to handle the CVC Lite array inference rules. The total HOL-Light code needed to use the CVC Lite theory is eight lines of definitions and two new theorems. This gives us all the power of a HOL-Light array theory with minimal effort.

4.1 Theory

The theory is a simple extensional theory of arrays, as found in [15]. There are two function symbols, **read** and **write**. The theory is axiomatized as follows:

$$(i = j \rightarrow (\mathbf{read}(\mathbf{write}\ a\ i\ v)\ j) = v)$$

$$(i \neq j \rightarrow (\mathbf{read}(\mathbf{write}\ a\ i\ v)\ j) = (\mathbf{read}\ a\ j))$$

$$((\forall i.\ (\mathbf{read}\ a\ i) = (\mathbf{read}\ b\ i)) \rightarrow a = b)$$

4.2 Results

Consider the following formula, where a and b are arrays:

$$(a = b) \rightarrow (\mathbf{write}\ a\ i\ (\mathbf{read}\ b\ i) = a)$$

Given the axioms, the built-in HOL-Light first order reasoner can solve this problem in 56 seconds. `CVC_PROVE` takes .015 seconds. Even slightly more difficult problems such as the following are intractable for HOL-Light. By contrast, `CVC_PROVE` solved it in 7.6 seconds.

$$(((\mathbf{write}\ a\ i\ v = \mathbf{write}\ b\ j\ w) \wedge (\mathbf{read}\ a\ i = v) \wedge (\mathbf{read}\ b\ j = w)) \rightarrow$$

$$((a = b) \wedge ((i = j) \rightarrow (v = w)) \wedge ((i \neq j) \rightarrow \mathbf{read}\ a\ j = w))$$

5 Future Research

There is an extensive proof theoretic literature on proof compaction. None of this is currently applied to the CVC Lite proofs. Also, for larger problems, it may be necessary to translate proofs in pieces to allow the entire object to fit in memory.

Another direction would be to allow more interaction between the two provers. Currently, there is no exchange of information other than a single proof. One possibility would be to allow the HOL-Light decision procedures

to call CVC Lite automatically when trying to solve subgoals during proof search.

6 Conclusion

This work demonstrates several benefits that can be derived from combining theorem provers. We presented concrete examples of a qualitative increase in the power of HOL-Light by translating proofs from CVC Lite. On the other hand, we also found an example where optimizing a decision procedure directly in HOL Light was superior to importing proofs from CVC Lite. Thus, we are forced to consider weather the gain in power which comes from using other tools is worth the effort.

In the Boolean satisfiability case, the combination is an unquestionable success. A good deal of the effort in improving SAT solvers has come from using advanced heuristics and data structures to enable a faster search through the Boolean search space. Imitating that in a system like HOL Light would take considerable effort and would still suffer from the problem that the search itself would be slower than in a tool written in a lower-level language and optimized for speed. It seems to make sense to use a fast tool to find the proof and then translate the proof into HOL-Light.

In the case of real arithmetic, where there is little search and the actual algorithm is very similar, optimizing directly in HOL Light was a better solution.

In the case of arrays, a theory which did not exist in HOL Light, the existence of a decision procedure in CVC Lite was easy and immediately useful.

One goal of combining theorem provers is to provide value in new contexts without having to duplicate human effort. This was amply demonstrated by the first and third examples. We hope that that this work will inspire additional efforts to combine theorem provers.

7 Acknowledgments

We'd like to thank New York University, the University of Pittsburgh, Carnegie Mellon University, and the National Science Foundation (CCR-ITR-0325808) for their support of this work. We'd also like to thank the referees for their helpful comments.

References

[1] http://www.logosphere.org.

[2] http://www.ags.uni-sb.de/~omega/.

[3] http://www-unix.mcs.anl.gov/AR/otter/.

[4] http://hol.sourceforge.net/.

[5] C. Barrett and S. Berezin. CVC Lite: A new implementation of the cooperating validity checker. In R. Alur and D. 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.

[6] C. W. Barrett, D. L. Dill, and J. R. Levitt. Validity checking for combinations of theories with equality. In M. Srivas and A. Camilleri, editors, *Proceedings of the* 1^{st} *International Conference on Formal Methods In Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, Nov. 1996. Palo Alto, California.

[7] M. Gordon, R. Milner, and C. P. Wadsworth. Edinburgh LCF: A Mechanised Logic of Computation. In *Lecture Notes in Computer Science*, volume 78. Springer-Verlag, 1979.

[8] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. In *Journal of the Association for Computing Machinery (JACM)*, volume 40, pages 143–184, January 1993.

[9] J. Harrison. HOL light: A tutorial introduction. In M. Srivas and A. Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag, 1996. see http://www.cl.cam.ac.uk/users/jrh/hol-light.

[10] J. Harrison. Formal verification of floating point trigonometric functions. In W. A. Hunt and S. D. Johnson, editors, *Formal Methods in Computer-Aided Design: Third International Conference FMCAD 2000*, volume 1954 of *Lecture Notes in Computer Science*, pages 217–233. Springer-Verlag, 2000.

[11] J. Harrison and L. Théry. A Sceptic's Approach to Combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.

[12] W. A. Howard. The formulae-as-types notion of construction. pages 479–490. Academic Press, 1998.

[13] L. Paulson. Logic and Computation: Interactive Proof with Cambridge LCF. In *Cambridge Tracts in Theoretical Computer Science*, volume 2. Cambridge University Press, 1987.

[14] A. Stump, C. W. Barrett, and D. L. Dill. CVC: A cooperating validity checker. In E. Brinksma and K. G. 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.

[15] A. Stump, D. L. Dill, C. W. Barrett, and J. Levitt. A decision procedure for an extensional theory of arrays. In *Proceedings of the* 16^{th} *IEEE Symposium on Logic in Computer Science (LICS '01)*, pages 29–37. IEEE Computer Society, June 2001. Boston, Massachusetts.