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

Sean McLaughlin[1], Clark Barrett[2], and Yeting Ge[2]

[1] Department of Computer Science, Carnegie Mellon University
seanmcl@cmu.edu
[2] 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. This technique fundamentally expands the capabilities of HOL-Light while preserving soundness.

1 Introduction and History

It is generally difficult to share results between different theorem provers. There are projects underway to address these issues. One example 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 [12]. Another example is Intel's Forte3 system [2] which uses various technologies including a theorem prover similar to HOL-Light, a model checker, and CVC Lite, to produce proofs of microprocessor correctness. Yet another example is the Ωmega system [3] that can import proofs of some other systems (eg. TPS) and uses other theorem provers for proof search in certain domains (eg. Otter [4] for first order logic).

In the hope of encouraging further progress along these lines, this paper describes a case study in which proof terms generated by the automatic theorem prover CVC Lite are translated into corresponding proofs in the interactive theorem prover HOL-Light. We give an example of a class of problems that were solved with `CVC_PROVE` but were unsolvable otherwise. Other less successful examples are given for comparison. We then add an array theory to HOL-Light with the help of CVC Lite. Finally, we discuss potential applications and future work.

1.1 Previous Work

The technique described in this paper is not new. It can be classified as a "skeptic's" approach to using external oracles with interactive theorem provers. One example of a similar project is described by Harrison and Théry [16] combining HOL and Maple. Decision procedures that produce checkable certificates are especially interesting. Examples include linear and semidefinite programming [17].

There has also been an attempt to provide automatic support for the theory of finite maps in HOL as a set of packaged tactics. This has been described in [20] and [9]

1.2 HOL-Light

HOL-Light [13] is an interactive theorem prover descended from the LCF projects [10, 18] and the HOL4 theorem prover [5]. 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 [14]. 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.

1.3 CVC Lite

CVC Lite [7] is an automatic proof-producing theorem prover for decidable first order theories. It is derived from the SVC and CVC projects at Stanford University [8, 19]. 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[3]. Nodes are labelled 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. The implementation code can be found at ???XXX???.

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 a boolean and real types. Terms in CVC are first order, a sublanguage of HOL-Light's higher order object language. Thus translating terms is easy as well.

2.2 Translating Proofs

Translating proofs formed the heart of this experiment. A proof in CVC Lite is a *proof term* in the sense of the Curry-Howard Isomorphism [15]. There are two advantages to the CVC Lite approach. First, it hides the proof search process, so that details of the search procedure (e.g., back-jumping, clause-conflict generation) are irrelevant. Second, the proof itself corresponds directly to a simple functional program. The translation process can be seen as executing this "program" in HOL-Light.

As an illustration, we demonstrate a proof of the proposition 'x=x' in CVC Lite.

```
(iff–mp true (= x x)
  (proof–by–contradiction true
    (let–p ((assump1 (not true)))
      (iff–mp (not true) false assump1 (rewrite–not–true)))
  (iff-symm (= x x) true (rewrite–eq–refl x))))
```

[3] The only complication being variable binding.

The inference rules act as follows[4]

$$\frac{}{\Gamma \vdash \textbf{rewrite–eq–refl}\, x : (x = x) = \textbf{true}}$$

$$\frac{}{\Gamma \vdash \textbf{rewrite–not–true} : \textbf{not}\, \textbf{true} = \textbf{false}}$$

$$\frac{\Gamma \vdash M : t_1 = t_2 \quad \Delta \vdash N : t_1}{\Gamma \cup \Delta \vdash \textbf{iff–mp}\ t_1\ t_2\ M\ N : t_2}$$

$$\frac{\Gamma \vdash M : t_1 = t_2}{\Gamma \vdash \textbf{iff-symm}\ t_1\ t_2\ M : t_2 = t_1}$$

$$\frac{\textbf{not}\ t \vdash M : \textbf{false}}{\Gamma \vdash \textbf{proof–by–contradiction}\ t\ M : t}$$

Then $\textbf{rewrite–eq–refl}\, x : (x = x) = \textbf{true}$, so

$$(\textbf{iff-symm}\ (=\ x\ x)\ \ \textbf{true}\ (\textbf{rewrite–eq–refl}\, x)))) : \textbf{true} = (x = x).$$

Assuming the middle portion corresponds to a verbose proof of **true**, then by considering its inference rule, the outer **iff**–**mp** term can be seen to correspond to a proof of $x = x$.

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. As an example, to translate the step labelled **proof**–**by**–**contradiction**, we must define[5] a HOL-Light derived rule (an OCaml function) which, given a proof of **false** from the term $\neg t$, a proof of t is produced. That is, we implement the CVC rule **proof**–**by**–**contradiction** in HOL Light, appealing to its own primitive rules. Thus, we not only do typechecking of the CVC Proof term, we prove that each CVC rule application is valid in the HOL-Light context.

```
let CCONTR =
  let P = ‘P:bool‘ in
  let pth = TAUT ‘(~P ==> F) ==> P‘ in
  fun tm th ->
    try let tm’ = mk_neg tm in
        MP (INST [tm,P] pth) (DISCH tm’ th)
    with Failure _ -> failwith "CCONTR";;
```

[4] As usual, the expression below the bar represents the conclusion, the expressions above the bar represent antecedents, and the judgment $A : \tau$ means, A has type (is a proof of) τ

[5] The rule CCONTR was written by John Harrison and is a part of the HOL-Light system. The actual rules we defined are longer and less instructive.

If the reader is unfamiliar with the HOL-Light style, the crucial point is that we can define the rule in terms of previously defined rules of inference (here `MP, INST, TAUT, DISCH`). The rule `CONTR` is then an ML function which takes a term `tm` and a theorem `th` of type `tm ==> F`. It then combines the previously defined rules to return the theorem `tm`.

We have a similar 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 were no errors, the translation of the root proof node yields the desired HOL-Light theorem.

3 Results

HOL-Light and CVC Lite have two overlapping theories, those of real arithmetic and boolean satisfiability. These are the realms at which we aimed our translation mechanism in order to determine its relative effectiveness.

3.1 Satisfiability

Consider the following class of problems. You are given $n-1$ sets of n pigeon-holes, arranged in n rows of $n-1$ columns. Given that no column can contain more than one pigeon, find a contradiction to the assertion that each row can contain a pigeon. For instance, this translates, for $n = 3$ as

$$\begin{aligned}&((\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_5) \wedge \\ &(\neg x_3 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_4) \wedge \\ &(\neg x_2 \vee \neg x_6) \wedge (\neg x_4 \vee \neg x_6) \wedge \\ &(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6)) \rightarrow \mathbf{false}\,.\end{aligned}$$

This is a notoriously difficult class of problems for typical boolean satisfiability methods. The following table[6] 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.

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

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.

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 [6]. Roughly, an array is a polymorphic type with two type variables, one

corresponding to the indexing type, and the other corresponding to the value type. There are two constants, **read** and **write**. There are two axioms in the theory. One, the *axiom of extensionality* for arrays, saying that two arrays are equal if and only if they have the same elements. The second is a *read over write* axiom, giving a simple term reduction.

4.2 Results

Consider the following HOL-Light term, where $S1$ and $S2$ are arrays where both type variables are instantiated by the *real* type:

$$(S1 = S2) \Rightarrow (\textbf{write}\ \ S1\ i\ (\textbf{read}\ \ S2\ i) = S1)$$

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.

$$((\textbf{write}\ \ S1\ i\ v = \textbf{write}\ \ S2\ j\ w) \wedge (\textbf{read}\ \ S1\ i = v) \wedge (\textbf{read}\ \ S2\ j = w) \Rightarrow$$
$$((S1 = S2) \wedge ((i = j) \Rightarrow (v = w)) \wedge ((i \neq j) \Rightarrow \textbf{read}\ \ S1\ j = w))$$

5 Future Research

5.1 Proof Size Reduction

There is an extensive proof theoretic literature on proof compaction. None of this is currently applied to the CVC Lite proofs.

5.2 Increased Interaction

Observe that there is really no exchange of information between the two theorem provers except in examining the proof term. One possibility is 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 been exploiting data locality and other "nonlogical" hardware-related heuristics. Imitating that kind of optimization in a system like HOL Light, written in a language without even the capability for managing memory layout, is impractical if not impossible.

In the case of real arithmetic, where 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.

As we see it, the goal of combining theorem provers, of which this paper is just a tiny exercise, is primarily to avoid the duplication of human effort. This was amply demonstrated by the first and third examples. Accordingly, we feel that this type of work has a very real value to the theorem proving community.

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.intel.com/software/products/opensource/tools1/verification.
3. http://www.ags.uni-sb.de/%7Eomega/.
4. http://www-unix.mcs.anl.gov/AR/otter/.
5. http://hol.sourceforge.net/.
6. Clark W. Barrett Aaron Stump, David L. Dill and Jeremy Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *IEEE Symposium on Logic in Computer Science*, volume 16. IEEE Computer Society, 2001.
7. Clark Barrett and Sergey Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In *CAV*, 2004. To appear.
8. 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.
9. A. D. Gordon and D. Syme. Automating type soundness proofs via decision procedures and guided reductions. In *9th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, volume 2514 of *LNCS*, pages 418–434. Springer, 1998.
10. 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.
11. Thomas Hales. http://www.math.pitt.edu/thales/kepler98.
12. Robert Harper, Furio Honsell, and Gordon Plotkin. A Framework for Defining Logics. In *Journal of the Association for Computing Machinery (JACM)*, volume 40, pages 143–184, January 1993.
13. John Harrison. HOL light: A tutorial introduction. In Mandayam Srivas and Albert 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.
14. John Harrison. Formal verification of floating point trigonometric functions. In Warren A. Hunt and Steven 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.
15. W. A. Howard. The formulae-as-types notion of construction. pages 479–490.
16. Laaurent Théry John Harrison. A Sceptic's Approach to Combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.
17. Stephen Obua.
18. L. Paulson. Logic and Computation: Interactive Proof with Cambridge LCF. In *Cambridge Tracts in Theoretical Computer Science*, volume 2. Cambridge University Press, 1987.
19. 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.
20. Don Syme. *Declarative Theorem Proving for Operation Semantics*. PhD thesis, U. of Cambridge, 1998.