System Description: An Interface Between CLAM and HOL*

Konrad Slind[1], Mike Gordon[1], and Richard Boulton[2] and Alan Bundy[2]

[1] University of Cambridge Computer Laboratory
[2] Department of Artificial Intelligence, University of Edinburgh

Abstract. The CLAM proof planner has been interfaced to the HOL interactive theorem prover to provide the power of proof planning to people using HOL for formal verification, *etc.* The interface sends HOL goals to CLAM for planning and translates plans back into HOL tactics that solve the initial goals. The project homepage can be found at `http://www.cl.cam.ac.uk/Research/HVG/Clam.HOL/intro.html`.

1 Introduction

CLAM [3] is a proof planning system for Oyster, a tactic-based implementation of the constructive type theory of Martin-Löf. CLAM works by using formalized pre- and post-conditions of Oyster tactics as the basis of plan search. These specifications of tactics are called *methods*. The planning-level reasoning may abstract the object-level reasoning, giving proof planning a heuristic element. Soundness is nevertheless guaranteed since proofs are constructed solely by sound tactics. Due to the heuristic aspect, the tactic application may be unsuccessful but in practice this is very rare. Experience also shows that the search space for plans is often tractable: CLAM has been able to automatically plan many proofs. A particular emphasis of research has been inductive proofs.

HOL [5] is a general-purpose proof system for classical, higher-order predicate calculus; it has been used to formalize many areas of interest to computer scientists and mathematicians. The HOL system has been criticized on the basis that it does not provide a high level of proof automation. Such remarks are often based on ignorance, since the HOL system now provides powerful simplifiers, automatic first order provers (both tableaux and model elimination), a semi-decision procedure for a useful fragment of arithmetic, and a co-operating decision procedure mechanism [1]. However, HOL lacks automation for many important areas. A good case in point is *induction*. Induction is certainly a central proof method, but in HOL, as in many other systems, the user must interactively control the application of induction.

* Research supported by the Engineering and Physical Sciences Research Council of Great Britain under grants GR/L03071 and GR/L14381.

C. Kirchner and H. Kirchner (Eds.): Automated Deduction, CADE-15
LNAI 1421, pp. 134–138, 1998.

These two systems have been linked to make the inductive proof methods of CLAM available to users of HOL, and also to give CLAM users access to the large libraries of tactics and theories available in HOL. CLAM is currently implemented in Prolog and HOL in Standard ML.

2 The Interface

In the current design, the HOL process is in control, treating CLAM as an intelligent remote tactic. The CLAM and HOL processes communicate over sockets. The sequence of operations is illustrated in Figure 1.

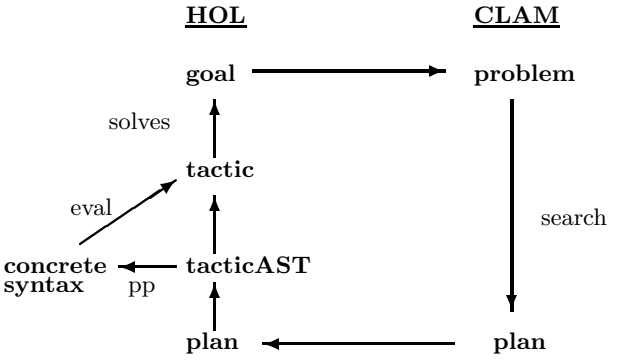

Fig. 1. System Structure

First, the HOL formula (goal) to be proved is translated into the syntax of Oyster's logic. This is then passed to CLAM, which searches for a proof. CLAM returns either a proof plan or an indication of failure. Supporting definitions, induction schemes, and lemmas are passed from HOL to CLAM in a similar way, prior to any proof attempts. CLAM acknowledges them with a handshaking message.

For successful proof attempts HOL receives a proof plan which it attempts to translate into corresponding tactics. If this is successful — which it normally is — the compound tactic is applied to the original HOL goal. Somewhat surprisingly, the plans CLAM produces based on specifications of Oyster tactics also work for a suite of purpose-written HOL tactics, despite the differences between the Oyster and HOL logics. These differences (and the heuristic aspects of planning) cannot lead to a non-theorem being 'proved' in HOL because HOL invokes its own tactics in checking the proof. Simply accepting as a theorem any goal for which CLAM finds a plan would be unsound.

3 Translation of the Object Language

In our work, CLAM has been modified to provide some independence from Oyster and the built-in types and induction schemes of the CLAM library. The library

mechanism and Oyster's well-formedness checks have been bypassed. This allows any HOL types and definitions to be used without having to make corresponding definitions in Oyster. Modification of CLAM to suit the classical higher-order logic used by the HOL system has largely been avoided by exploiting correspondences between syntactic features of HOL's logic and the constructive type theory of Oyster/CLAM. The Oyster type theory has not been changed and no new methods have been written, though one for classical propositional reasoning might assist in some examples.

The HOL logic is translated to the syntax used by CLAM as follows. **F** (falsehood) is translated to the empty type and **T** (truth) to the type used to represent truth in CLAM. Conjunction translates to a product type, disjunction to a disjoint union type, implication to a function type, and negation to a function type between the argument of the negation and the empty type. Quantifications become dependent types. Equality between booleans is translated to if-and-only-if and other HOL equalities become equalities in CLAM. Other HOL terms are translated almost directly into the corresponding type-theoretic constructs.

Types in HOL are distinct from formulas/terms and so are translated separately. This is largely straightforward, although type variables required some thought. In HOL, type variables are implicitly universally quantified, but in CLAM they have to be bound. Thus, at the top level, the variables introduced for HOL type variables are quantified by placing them in the first type universe, **u(1)**. As Felty and Howe [4] point out, the domain should really be restricted to the *inhabited* types of **u(1)** since HOL types have to be non-empty. However, for the kinds of proof under consideration this will be of no consequence and as pointed out earlier can not lead to inconsistency in HOL.

Differences in the lexical conventions of the HOL logic and those of CLAM (which are essentially those of Prolog) require some translation of constant and variable names. The translation table is retained for use in translating the proof plan to a HOL tactic.

4 Tactic Generation

A proof plan is translated into a composition of 'atomic' tactics in HOL, each of which corresponds to a method of CLAM. Currently, there are about twelve atomic tactics that form the basis of the translation.

Tactic generation takes place in two stages, as can be seen in Figure 1. First, an abstract syntax representation (**tacticAST**) of the tactic is derived from the plan. The abstract syntax is then used to generate either a tactic (an ML function) for direct application to the goal or a textual representation (ML concrete syntax) of the tactic for inclusion in a file. Direct translation into a tactic allows the plan to be applied to the goal without parsing and evaluating ML code. On the other hand, the generation of concrete syntax (by **pp**) allows the tactic to be inserted in ML tactic scripts and used in HOL sessions where CLAM may not be present, *i.e.*, it provides persistence.

One of the challenges in translating plans to tactics is tracking (in HOL) the variables introduced into the proof by CLAM. For example, consider an induction step in a plan: the step cases (and sometimes the base cases) introduce new bound variables. Later in the plan, these variables may become free as a result of specialization. Still later, a term with such free variables may be generalized. For HOL to make the same generalization step, the HOL goal must have corresponding occurrences of the same term (and hence corresponding occurrences of the same free variables). Therefore the proof plan must provide sufficient information for the names of bound variables in induction schemes to be ascertained.

5 Examples Performed

Examples that have been planned by CLAM and proved in HOL using the interface include commutativity of multiplication (over the naturals) and a number of theorems about lists including some known to be difficult to automate. The interest in many of these examples is not primarily the theorem, which is usually fairly simple, but rather in how CLAM found the proof, by making multiple and nested inductions and generalizations. Here are a few concrete examples:

$\forall x\ y.\ \texttt{REVERSE}\ (\texttt{APPEND}\ x\ y) = \texttt{APPEND}\ (\texttt{REVERSE}\ y)\ (\texttt{REVERSE}\ x)$
$\forall x\ m\ n.\ \texttt{APPEND}\ (\texttt{REPLICATE}\ x\ m)\ (\texttt{REPLICATE}\ x\ n) = \texttt{REPLICATE}\ x\ (m + n)$
$\forall x\ m\ n.\ \texttt{FLAT}\ (\texttt{REPLICATE}\ (\texttt{REPLICATE}\ x\ n)\ m) = \texttt{REPLICATE}\ x\ (m * n)$

The functions here are curried. APPEND concatenates two lists, REVERSE reverses a list, FLAT flattens a list of lists into one list (by iterated concatenation), and REPLICATE x n generates a list of n copies of x.

6 Conclusions

Two mechanized reasoning systems, one interactive with a large library of theories and many significant examples (HOL), and the other a largely automatic prover (CLAM), have been connected to provide a potentially useful tool for formal verification. The inductive methods of CLAM complement existing proof tools in HOL. Although the system is still very much a prototype, early results are promising. A more detailed, though less up-to-date, description of the system is available as a technical report [2]. Future goals include extending the range of formulas handled, more extended interaction between the two systems (*e.g.*, recursive dialogues), and testing on medium to large examples.

References

1. R. J. Boulton. Combining decision procedures in the HOL system. In *Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, volume 971 of *Lecture Notes in Computer Science*. Springer, 1995.
2. R. Boulton, A. Bundy, K. Slind, and M. Gordon. A Prototype Interface between CLAM and HOL. Research Paper 854, Department of Artificial Intelligence, University of Edinburgh, June 1997.
3. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7(3):303–324, 1991.
4. A. P. Felty and D. J. Howe. Hybrid interactive theorem proving using Nuprl and HOL. In *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, volume 1249 of *Lecture Notes in Artificial Intelligence*. Springer, 1997.
5. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.