System Description: An Interface between CLAM and HOL*

Konrad Slind[1], Mike Gordon[1], Richard Boulton[2], 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 [2] 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*. When a plan for a goal is found, the expectation is that the resulting tactic will solve the goal. Experience shows that the search space for plans is often tractable: CLAM has been able to automatically plan many proofs. A particular emphasis of research with CLAM has been the automation of 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. However, HOL lacks automation for many important areas, and moreover, there is always more that can be done to automate the proof process. 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.

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 for HOL. CLAM is currently implemented in Prolog and HOL in Standard ML.

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

2 The Interface

The CLAM and HOL processes communicate over sockets. These may be either local file-system sockets or socket connections over the Internet. In the current set-up, the HOL process is in control, using CLAM as an intelligent remote tactic. The sequence of operations is illustrated in Figure 1.

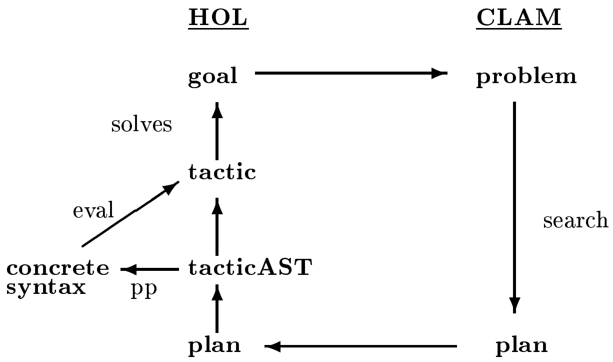

Fig. 1. System Structure

First, the HOL formula (goal) to be proved is translated into the abstract syntax of Oyster's logic. This is then written to the socket (in concrete syntax) as a Prolog goal. The CLAM process waits for a message from HOL and, on receiving one it recognizes, executes it and either returns the result back down the socket or sends a handshaking message. Supporting definitions, induction schemes, and lemmas are passed from HOL to CLAM in a similar way, prior to any proof attempts.

For successful proof attempts HOL receives a proof plan (again in concrete syntax) back from CLAM. HOL then parses the plan and attempts to translate it into a corresponding tactic. If this is successful (which it normally is) the tactic is applied to the original HOL goal. Since CLAM uses heuristics the tactic application may be unsuccessful but in practice it is very rare for CLAM to return an inappropriate plan. Most importantly, an inappropriate plan can not lead to a non-theorem being 'proved' in HOL because HOL invokes its own tactics (guided by the plan) in checking the proof.

Tactic generation takes place in two stages. First, an abstract syntax representation of the tactic is generated. This is where most of the work is. The abstract syntax is then used to generate either a tactic function (an ML function) for direct application to the goal or a textual representation of the tactic for inclusion in a file. Translation into a tactic function (*i.e.*, the internal representation) allows the plan to be applied to the goal without parsing and evaluating ML code. The generation of concrete syntax (by pretty-printing) allows the tactic to be inserted in ML tactic scripts and used in other HOL sessions. The abstract syntax for tactics carries both a pretty-printer and a tactic at each leaf node, so that compound concrete syntax and tactics can be generated easily.

3 Translation of the Object Language

The CLAM process used has been modified to provide some independence from Oyster and the built-in types and induction schemes of the CLAM library. 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 HOL logic is translated to the syntax used by CLAM as follows. False is translated to the empty type and true to the special type used to represent true in CLAM. Conjunction is translated 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. Universal quantification becomes a dependent function and existential quantification a dependent product. Equality between booleans is translated to if-and-only-if and other HOL equalities to equalities in CLAM. Decidability issues for the latter create some problems in planning. 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 (in a straightforward manner).

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 reverse translating the proof plan to a HOL tactic.

In HOL, type variables are implicitly universally quantified. In CLAM they have to be bound. So, at the top level, the variables introduced for HOL type variables are quantified by assuming that they inhabit the first type universe, **u(1)**. As Felty and Howe [3] 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.

4 Translation of Plans to Tactics

Tactics are a well-known method for backward proof. The original conception of Milner [4], which is still that of tactics in HOL, is that a tactic can be represented by the type $goal \longrightarrow goal\ list * justification$, *i.e.*, a tactic decomposes a goal into subgoals plus a justification function. The justification function takes the theorems resulting from the solved subgoals and performs inference with them to return a new theorem that *achieves* the original goal. Thus the justification has type $thm\ list \longrightarrow thm$. Currently the interface with CLAM assumes that the list of subgoals is empty, *i.e.*, the plan completes the proof.

New HOL tactics have been implemented that correspond to the low-level methods used by CLAM. One of the challenges in maintaining the correspondence between tactics and methods is tracking in HOL the variable names used by CLAM. This is because of generalization: when CLAM generalizes a goal — a step that the translation must track — it does so with an explicit term, which can have occurrences of variables from induction templates. For HOL to make the same step, its goal must have corresponding occurrences of the term. This information must be extractable from the proof plan for the translation to work.

5 Examples Performed

Examples that have been planned by CLAM and proved in HOL using the interface include the commutativity of multiplication (over natural numbers) 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 useful tool for formal verification. The inductive methods of CLAM complement existing proof tools in HOL, *e.g.*, Boulton's co-operating decision procedure package [1]. Although the system is still very much a prototype, early results are promising. 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. 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.
3. 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.
4. M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.
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.