The OYSTER-CLAM system

The OYSTER-CLAM system *

Alan Bundy Frank van Harmelen Christian Horn†
Alan Smaill
Department of Artificial Intelligence
University of Edinburgh

1 OYSTER

OYSTER [Hor88] is an interactive proof editor closely based on the Cornell NuPRL system, but implemented in Prolog. The object-level logic is a version of Martin-Löf type theory (a higher order constructive logic including induction) in a sequent-calculus formulation. Proofs are constructed in a top-down fashion by application of the rules of inference. Notational definitions and libraries of theorems are supported.

The tactic language for the system is Prolog. Predicates describing properties of a proof under construction are available to the user, who may also include arbitrary Prolog in tactics. Soundness of the system is ensured by the use of an abstract data type of proofs: partial proofs can only be altered by application of the primitive proof rules. Tactics can be combined using system defined tacticals. Prolog pattern-matching and backtracking in tactics have proved useful in the automation of proof search.

Since the object-level logic is constructive, terms of an enlarged λ-calculus can be computed from complete proofs, and these so-called *extract terms* can then be executed by application on appropriate inputs. This allows the system to be used as a program synthesis environment, since a theorem can be regarded as a specification which is realised by its extract term.

The system is written in some 2000 lines of Prolog, making it considerably more compact than the original NuPRL system, while the speed of the two systems is comparable.

*The research reported in this paper was supported by SERC grant GR/E/44598, and an SERC Senior Fellowship to the first author.

†Author's address: Department of Mathematics, Humboldt University, Postfach 1297, 1086 Berlin, GDR

2 CLAM

CLAM is a meta-level system built on top of OYSTER to turn the interactive proof editor into a fully automatic theorem proving system.

For every tactic written in the OYSTER system, CLAM is equipped with a specification of the tactic. We call such specifications *methods*. Methods consist of an input formula, preconditions, output formulae and postconditions. A method is said to be *applicable* if the goal to be proved matches the input formula of a method, and the method's preconditions hold. The preconditions, formulated in a meta-logic, refer to syntactic properties of the input formula. Using the input formula and preconditions of a method, CLAM can predict if a particular OYSTER tactic will be applicable without actually running it. Similarly, the output formulae gives a *schematic* description of the formulae resulting from a tactic application, and the postconditions specify further syntactic properties of these formulae.

CLAM employs these methods in the search for a proof of a given formula by finding an applicable method, computing the schematic output formulae and postconditions of this method, and finally finding methods applicable to these output formulae, this process being repeated recursively until no unproved formulae remain.

We call this process of concatenating methods by search at the meta-level *proof planning*, and the resulting tree of methods is called a *proof plan*. This proof plan can then be executed at the object-level: for each method in the proof plan, the system will execute the corresponding OYSTER tactic. This process of plan execution is not guaranteed to succeed, though it typically does. Thus the method acts as a heuristic operator which can capture the essential preconditions of a tactic while leaving out expensive checks for finer details.

The process of proof plan construction as described above involves search: for a given sequent, more than one method may be applicable. We have experimented with a number of different search strategies (depth-first, breadth-first, iterative deepening, heuristic search). In practice, the search at the meta-level is small enough for a depth-first planner to succeed without very much backtracking, often none.

CLAM is implemented in 2300 lines of Prolog code, and currently contains a set of methods and tactics modelled on the Boyer-Moore theorem prover. The system can, for instance, find an automatic proof of the existence of prime factorisation, thus extending the work by Boyer and Moore, since CLAM not only verifies a given algorithm for prime factorisation, but indeed synthesizes this algorithm from an existentially quantified specification. CLAM takes 176 seconds to construct a plan for this theorem consisting of 15 methods, while the object-level proof consists of 553 steps, executed by OYSTER in 358 secs. More detailed descriptions of CLAM can be found in [vH89] and [BvHS89].

References

[BvHS89] A. Bundy, F. van Harmelen, and A. Smaill. Extensions to the rippling-out tactic for guiding inductive proofs. Research Paper 459, Dept. of Artificial Intelligence, University of Edinburgh, 1989. Also in the proceedings of CADE-10.

[Hor88] C. Horn. The Nurprl proof development system. Working paper 214, Dept. of Artificial Intelligence, University of Edinburgh, 1988. The Edinburgh version of Nurprl has been renamed Oyster.

[vH89] F. van Harmelen. The CLAM proof planner, user manual and programmer manual: version 1.4. Technical Paper TP-4, DAI, 1989.