Continuations of Proof Strategies

Julian Richardson

Department of Computing and Electrical Engineering
Heriot-Watt University, Edinburgh EH14 4AS, Scotland
julianr@cee.hw.ac.uk

In this paper we define a notion of continuation *for proof planning strategies which allows us to represent explicitly the progress of proof plan construction, define a planner which allows planning to be interrupted, and prove that this planner is equivalent to the standard proof planner.*

Proof planning [Bun88] is an automated theorem proving technique which builds a proof in two steps: first, a *proof plan* is constructed by applying plan operators (called *methods*) which encode high-level proof steps to a plan state which is initially the conjecture to be proved and in general consists of subgoals of the theorem proving process. Second, when a complete proof plan has been constructed, it translated into a composition of tactics which may be applied to the conjecture in order to construct a concrete fully formal proof.

Complex proof strategies could be defined in the first proof planner, *Clam* [BvHHS90], by allowing methods to explicitly invoke other methods. More recently, the higher-order proof planner λ*Clam* [RSG98] defined a language for plan expressions which has explicit control structures called *methodicals*, analogous to the *tacticals* of tactic-based theorem proving: *then_meth*, *repeat_meth*, *orelse_meth* etc. These plan expressions are used to define proof strategies. For example the standard proof strategy for inductive proof is as follows:

```
method_def top_meth
  (complete_meth
    (repeat_meth
      (orelse_meth all2i
        (orelse_meth taut
          (orelse_meth sym_eval
            (orelse_meth existential
              (orelse_meth (repeat_meth generalise)
                             ind_strat))))))).
```

`Taut`, `sym_eval` and `ind_strat` are themselves complex proof strategies, while `existential`, `generalise` are methods.

Methodicals are implemented as λProlog clauses for a plan construction predicate `apply/3` which applies the proof strategy specified in its first argument to the goal in its second argument producing the goal in its third argument. For example:

```
apply (then_meth M1 M2) Gin Gout :-
   apply M1 Gin Gmid, apply M2 Gmid Gout.

apply (orelse_meth G1 G2) M1 M2 :-
   apply G1 M1 M2.
```

```
apply (orelse_meth G1 G2) M1 M2 :-
   apply G2 M1 M2.
```

Proof plans are then constructed by interpreting proof strategies using these clauses (amongst others) in a depth-first manner. Backtracking may occur. Such interpretation is efficient, but is inflexible; for example, we cannot stop the planning process and continue later because the point we have reached in the plan construction process is implicitly held on the λProlog execution stack. Representing explicitly the state of proof plan construction makes it possible to pause the search for a proof plan, and navigate or modify the current proof state in an interactive proof planner such as *XBarnacle*. It also allows proof critics [Ire92] to analyse the proof state after a failure.

Continuations [Rey93] have been used in computer science for many years to reason about control structures in programs. We have defined a notion of continuation for proof strategies which makes proof states explicit by defining a transformation on proof strategies which is guaranteed to transform any complex proof strategy S into a form *then_meth* F T where F is a method and T is the continuation — another complex proof strategy which should be applied to the proof state arising from the execution the method F such that the applications of S and of *then_meth* F T are equivalent. We have defined a proof planner which uses continuations and permits proof search to be paused, modified and resumed, and allows reasoning about proof states. We can prove that it is equivalent to the standard depth-first proof planner. This proof planner has been implemented in λ*Clam*.

Continuations may be useful in the future for reasoning about proof planning strategies, providing them with semantics and transformations in a way akin to their fruitful application to programming languages.

References

[Bun88] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.

[BvHHS90] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

[Ire92] A. Ireland. The Use of Planning Critics in Mechanizing Inductive Proofs. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning – LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence No. 624, pages 178–189. Springer-Verlag, 1992. Also available from Edinburgh as DAI Research Paper 592.

[Rey93] J. C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6:233–247, 1993.

[RSG98] J.D.C Richardson, A. Smaill, and Ian Green. System description: proof planning in higher-order logic with λClam. In Claude Kirchner and Hélène Kirchner, editors, *15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, Lindau, Germany, July 1998.