



THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

Learning from Experts to Aid the Automation of Proof Search.

Citation for published version:

Bundy, A, Grov, G & Jones, CB 2009, Learning from Experts to Aid the Automation of Proof Search. in *Proceedings of the Ninth International Workshop on Automated Verification of Critical Systems (AVOCS 2009)*.

Link:

[Link to publication record in Edinburgh Research Explorer](#)

Document Version:

Early version, also known as pre-print

Published In:

Proceedings of the Ninth International Workshop on Automated Verification of Critical Systems (AVOCS 2009)

General rights

Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.





Proceedings of the
Ninth International Workshop on
Automated Verification of Critical Systems
(AVOCS 2009)

Learning from experts to aid the automation of proof search

Alan Bundy , Gudmund Grov and Cliff B. Jones

2 pages

Learning from experts to aid the automation of proof search

Alan Bundy¹, Gudmund Grov² and Cliff B. Jones³

¹ University of Edinburgh ² Heriot-Watt University ³ University of Newcastle

Abstract: Most formal methods give rise to proof obligations (POs) which are putative lemmas that need proof. Discharging these POs can become a bottleneck in the use of formal methods in practical applications. It is our aim to increase the repertoire of techniques for reducing this bottleneck by tackling learning from proof attempts. In many cases where a correct PO has not been discharged, an expert can easily see how to complete a proof. We believe that it would be acceptable to rely on such expert intervention to do one proof if this would enable a system to kill off others “of the same form”.

Keywords: Formal verification, automated theorem proving, proof planning

Our objective is to decrease significantly the human effort involved in doing top-down formal development. We believe this can be achieved by increasing the proportion of POs that are discharged by formal method support systems. Specifically, we hope to build a system that will learn enough from one proof attempt to improve the chances of proving “similar” results automatically.¹ By “proof attempt” we include things like the order of the steps explored by the user² (not just the chain of steps in the final proof). Thus it is central to our goal that we find *high-level* strategies capable of cutting down the search space in proofs. What we are looking for is at a much higher level than LCF-style tactic languages — such tactics are programs to construct proofs and are brittle in the sense that they behave differently for similar POs.

By separating information about data structures and approaches to different patterns of POs, a taxonomy begins to evolve. A proof (attempt) might be seen to use “generalise induction hypothesis” (e.g. adding an argument to accumulate values) in a specific proof about, say, sequences; a future use of the same PO might involve a more complicated tree data structure — but if it has an extended induction rule, the same strategy might work.

So our hypothesis is: *we believe that it is possible to (devise a high-level strategy language for proofs and) extract strategies from successful proofs that will facilitate automatic proofs of related POs.*

Designing the strategy language is part of a proposed project but it might be useful to give some indication of what we expect it to look like. Our strategy language will combine a high-level proof strategy with a “vocabulary” of terms that might be instantiated in the separate theories of data structures stored in the system. The meta-language employed in our rippling/induction proof-planning work [BBHI05] provides an existence proof for such a strategy language. Items that we expect to play a major part include:

¹ At the first review of the EU-funded DEPLOY project (<http://www.deploy-project.eu>), one of the industrial partners reported on an application that gave rise to some 300 POs; of these, about 200 were discharged automatically; five really difficult proofs were done by hand and although the remaining 95 “followed the pattern” of the five, they also had to be done slowly and manually.

² We envisage a style of flexible interaction like that from “mural” [JLM91].



Some ‘standard’ proof plans and known deviations from and patches to them. [BBHI05] uses rippling to describe a ‘standard’ proof plan for inductive proofs and shows how each different pattern of failure in rippling suggests a different way of patching a failed proof attempt. We hypothesise that expert-provided proofs of undischarged POs will typically exhibit either a new proof plan or a new patch to an existing plan.

Choices of unusual induction rules and variables, choices of loop invariants. Choosing an alternative non-standard induction rule is one of the patches to the standard induction proof plan which is described in [BBHI05]; patching a failed loop invariant is the patch described in [SI98].

Choices of intermediate lemmas. Designing, constructing and proving a key intermediate lemma is another of the patches to the standard induction proof plan described in [BBHI05].

Generalisation of the PO. [BBHI05] also describes ways of generalising the PO or the current goal to patch a failed proof.

Lemmas, case splits, loop invariants, generalisations and their points of application all need to be described in an abstract form if they are to apply to all members of a family of proofs. This is because the details will vary from proof to proof, but there may be a level of abstraction at which their descriptions coincide. Rippling, for instance, provides an exemplar abstract language, since it can describe ‘missing’ intermediate lemmas in terms of subexpressions that must match with different parts of the current goal. We will also make use of generic taxonomies, for instance, types of induction rule, types of generalisation, etc. to support the abstraction of proofs and their subsequent application to new conjectures. For instance, the use of a two-step induction on a recursive data-structure in the source proof must first be abstracted in order to be applied to a different data-structure in the target proof. We expect to develop and use additional kinds of abstraction during the course of the project.

The major challenge is to design a sufficiently general-purpose and robust strategy language so that it can deal with unanticipated proof plans and patches that experts will devise. If we knew in advance what these plans and patches would be, we could include them in the theorem prover, so that the problematic POs would be discharged and would not require expert attention.

Bibliography

- [BBHI05] A. Bundy, D. Basin, D. Hutter, A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning*. Cambridge Tracts in Theoretical Computer Science 56. Cambridge University Press, 2005.
- [JLM91] C. B. Jones, K. D. Jones, P. A. Lindsay, R. Moore. *mural: A Formal Development Support System*. Springer-Verlag, 1991.
<http://homepages.cs.ncl.ac.uk/cliff.jones/ftp-stuff/mural.pdf>
- [SI98] J. Stark, A. Ireland. Invariant Discovery via Failed Proof Attempts. In Flener (ed.), *Logic-based Program Synthesis and Transformation*. LNCS 1559, pp. 271–288. Springer-Verlag, 1998.