

AI4FM

A new project seeking challenges!

Gudmund Grov¹ and Cliff B Jones²

¹ School of Informatics, University of Edinburgh, UK,
ggrov@inf.ed.ac.uk

² School of Computing Science, Newcastle University, UK,
cliff.jones@ncl.ac.uk

Abstract. The “proof obligations” generated from many formal methods tend to be simple and can often be discharged by modern automatic theorem provers or SMT systems. However, those proof tasks that need hand –or interactive– intervention present a barrier to the use of formal methods. Theorem proving was one of the earliest challenges addressed by researchers in the area of Artificial Intelligence and enormous progress has been made in the provision of general purpose heuristics. The approach in the recently started AI4FM project is different: we hope to devise a system that will learn from an expert user how they tackle one interactive proof and then apply the discovered high-level strategy to other related proof tasks. We are fortunate in having access to many such problems through the DEPLOY project but are aware of the dangers of devising an overly specific approach. This short paper appeals for challenge problems from other sources.

1 Introduction

We have *just* embarked on a four-year research project (AI4FM) that will use Artificial Intelligence (AI) to tackle a core issue for “Formal Methods”³ and we are keen to receive challenge problems.

Achieving verified software has been a dream since the birth of computer science [Jon03] and the importance of this objective has become ever greater with the increasing size and complexity of software.⁴

The use of formal methods has been successful in safety-critical domains, such as railway and aviation; gradually they are becoming increasingly popular in other sectors (e.g. Microsoft use formal methods to verify device drivers). A recent paper by Woodcock *et al.* [WLB09] provides an up-to-date analysis of a significant number of industrial applications of formal methods. As the use of formal methods has spread beyond small groups of experts out to far larger

³ ‘Formal methods’ use mathematics to specify, develop and reason about software and systems.

⁴ We use the term software although the discussion here is valid for any application of formal methods, i.e. to generic system modelling as well as both hardware and software.

groups of industrial engineers, the importance of the availability of various sorts of “support tools” has been recognised. Such support tools include parsers/type checkers, full-blown theorem provers and decision procedures.

Formal methods are applied both *post facto* and in VxC *verified by construction*. As a shorthand, the former are often referred to as “bottom up” and the latter as “top down”. Both approaches have their place and our decision to focus our current efforts on VxC has more to do with the applications that interest us than in any value judgement.

Top-down methods such as VDM [Jon90], B[Abr96] or Event-B[Abr10] tend to subscribe to a “posit and prove” pattern in which a designer posits a step of development and then seeks to justify it. By focusing on a particular style of development, a support tool can be built that generates “proof obligations” (POs) whose discharge justifies the correctness of a development step. For Event-B, one such support system is known as the “Rodin Tools”⁵— the generated POs are putative lemmas that need proof. In a carefully structured Event-B development, the theorem provers in the Rodin Tools will discharge the majority of the POs automatically; with skill and experience, users can get the percentage discharged automatically into the nineties. These results are typical of other choices of methods/tools. The remaining POs need to be discharged by interactive proof and this can be both time-consuming and challenging for industrial users; this in turn leads to proof being a bottleneck in industrial deployment.

There are, in fact, two approaches to dealing with the POs that require user interaction:

1. Follow a *modelling strategy*: change the model/abstraction in order to simplify the proofs, thus increasing the proportion of POs that are discharged automatically. For example, in [BY08], extra refinement steps are introduced due to known limitations of the automatic theorem provers.
2. Follow a *proof strategy*: accept the challenging POs and define a strategy for discharging them.

Both approaches are valid and useful and they can be seen as complementary. For example, the numbers quoted below most likely apply after several iterations of massaging the original model. A proof strategy could still be applied after the modelling strategy has reduced the numbers of undischarged POs.

It is the proof approach that we will take first in our AI4FM project. Our principal aim is to increase the repertoire of techniques for the proof-strategy approach by learning from proof attempts made by humans. We should make it clear that we are thinking of higher-level strategies than those normally coded in say HOL or Isabelle. In fact our model of the proof process is more like the sort of interactive proofs created in [JJLM91]; we are also aware that the progress in SMT research will have an influence on our approach. For example, we may incorporate SMT solvers as new tactics.

It should be remembered that the POs arising from formal methods tend to have different properties from “pure” mathematics.

⁵ See www.event-b.org

1. There are often large numbers of detailed POs. To illustrate, the Paris Metro Line 14 and the Roissy Airport shuttle system were both developed using B [Abr07]; the former generated 27,800 POs (around 2,250 interactive) while the latter generated 43,610 POs (around 1,150 interactive).
2. POs tend to be less deep.
3. They often exhibit a “similarity”, in the sense that they can be grouped into “families” and the same (high-level) proof approach can be successfully applied to all members of the family.

2 The AI4FM approach

There are two reasons of why a PO might not be discharged automatically: the putative lemma could be wrong (thus pinpointing a mistaken design decision⁶) or a true lemma has not been proved by the theorem prover because it is “not smart enough”.

We have some data from industrial use that suggests failing POs fall into a relatively small number of distinct classes in the sense that one new idea will be key to discharging many POs. It is tempting to search for ever better heuristics but we plan to follow a different path in AI4FM. In many cases where a (correct) PO is not discharged automatically, an expert can easily see how to complete a proof. By exploring the nature of the POs within formal methods we believe that a higher degree of automation can be achieved by relying on expert intervention to do one proof, with the expectation that this would enable the system to discharge other POs in the same family.

Specifically, we hope to build a tool that will learn enough from one proof attempt to improve the chances of proving “similar” results automatically. By “proof attempt” we include things like the steps explored by the user (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.

Our hypothesis is:

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

To achieve our goal we plan to analyse exemplar proofs (including their starting PO) using many *dimensions*. For example, we might separate information about data structures and approaches to different patterns arising from POs. Thus one proof (attempt) might be seen to use “generalise induction hypothesis” (e.g. adding an argument to accumulate values) about, say, sequences; a future use might involve a more complicated tree data structure — but if it has an induction rule, the same strategy might work. We hope to pick out other dimensions, such as the domain of the application that gave rise to the PO (e.g. does it relate to trains or to railway tracks?).

⁶ An AI approach to help with these circumstances is discussed in [IGB10]

Designing a strategy language capable of capturing such properties (in an abstract form) is crucial to the success of AI4FM — one indication of the feasibility of such an approach is the earlier work on “proof critics” and “rippling” [BBHI05] — some early thoughts on a strategy language are presented in [JGB10] — and some *simple* examples in [BGJ09].

A key question for the design of the strategy language is the level of abstraction that will be used. We see two extreme points:

- a rather concrete description of a proof strategy (close to the tactic level), would not require much proof search when re-applying the strategy on POs in the same family – however, the size of the family would be rather small;
- much more abstract descriptions of proof strategies would capture far broader families of POs – however, they would require more proof search.

Thus, to reduce proof search, whilst keeping the families large, a language enabling proof strategy descriptions at different levels of abstractions seems desirable. HiProofs [DPT06] is an example to describe tactic proof using many levels, and we plan to build on this idea.

We would also need to provide tool support to both extract strategies from an exemplar interactive proof – and to interpret a strategy to discharge POs in the same family as the exemplar proof. We plan to build, at least the interpreter, on top of the Isabelle proof assistant. The advantage of Isabelle, is that it contains a meta-logic where we can, to some degree, develop our system – independent of the underlying method and logic.

Our solutions will rely heavily on heuristics — we do not believe that there are algorithmic solutions to most of our problems. Thus, as the project name suggests, our techniques will be heavily influenced by artificial intelligence. Particular areas of artificial intelligence we hope can help are

- *planning* and *proof planning* to find proofs from strategies – e.g. as in rippling discussed above;
- *machine learning* in order to
 - extract strategies from exemplar proofs. For example, *Explanation Based Generalisation/Learning* has previously been used to generalise sub-proofs for reuse [MS98]. However, something more general is probably required for our purposes;
 - discovering related POs, or finding a particular strategy for a sub-proof of a PO. We will need to use a form of pattern recognition in order to achieve this – which explores the various dimensions as discussed above.
- from the exemplar proof we may find “dead ends” in the search space, and use *search techniques* to rule them out from the target search space – i.e. strategies at the level of the search space.

3 We need more challenge portfolios

We are fortunate that our access through the DEPLOY project⁷ to industrial users of the Rodin support tools will facilitate the capture of many difficult POs.

⁷ See www.deploy-project.eu

We have already begun to find out how easy it is to analyse them into families that succumb to similar ideas to get their proofs to go through. But we are aware that it is always dangerous to base research on too narrow a base. We should therefore like to elicit challenge problems from other projects.

We can see three levels of useful access.

- Simple: we would be interested to receive POs generated from formal models of non-trivial computer systems — if these are beyond the power of the automatic theorem provers and/or SMT systems at hand, they might be interesting challenges for us — we are aware that even transferring a single model is not a simple file because we will need any base (data type) theories and, potentially, information about the logic used
- Valuable: it would be even more useful if we could get access to families of related proofs — we would be quite happy to get part of set (from which we try to learn strategy) — and then subject to independent scrutiny the question of whether the strategies that we devise would help with the unseen proof tasks
- Optimal: if we could, in addition to the above, receive a proof history including “this is where our TP got stuck” we would gain more insight into what is needed

We would appreciate it if anyone considering sending us material contacted Gudmund Grov⁸ since dialogue is more likely to make the process work. We fully understand that industrial users might wish to disguise details of their models by changing the names of functions and/or state components.

Acknowledgements We would like to thank all members of the AI4FM project, in particular Alan Bundy. This work is supported by EPSRC grant (EP/H024204/1 and EP/H024050/1): ‘*AI4FM: the use of AI to automate proof search in Formal Methods*’.

References

- [Abr96] J.-R. Abrial. *The B-Book: Assigning programs to meanings*. Cambridge University Press, 1996.
- [Abr07] J.-R. Abrial. Formal methods: Theory becoming practice. *Journal of Universal Computer Science*, 13(5):619–628, 2007.
- [Abr10] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
- [BBHI05] A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning*, volume 56 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2005.
- [BGJ09] Alan Bundy, Gudmund Grov, and Cliff B. Jones. An outline of a proposed system that learns from experts how to discharge proof obligations automatically. In *Proceedings of Dagstuhl Seminar 09381: Refinement Based Methods for the Construction of Dependable Systems*, 2009.

⁸ See the AI4FM web page www.ai4fm.org or email ggrov@inf.ed.ac.uk

- [BY08] Michael Butler and Divakar Yadav. An Incremental Development of the Mondex System in Event-B. *Formal Aspect of Computing*, 20(1):61–77, 2008.
- [DPT06] Ewen Denney, John Power, and Konstantinos Tourlas. Hiproofs: A hierarchical notion of proof tree. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 155:341–359, 2006.
- [IGB10] Andrew Ireland, Gudmund Grov, and Michael Butler. Reasoned Modelling Critics: Turning Failed Proofs into Modelling Guidance. In *Proceedings of ABZ'10*, number 5977 in LNCS, pages 189–202. Springer-Verlag, 2010.
- [JGB10] Cliff B. Jones, Gudmund Grov, and Alan Bundy. Some facets of a strategy language for proofs. In *5th Automated Formal Methods workshop (AFM'10)*, July 2010. Also available as Edinburgh University, School of Informatics technical report EDI-INF-RR-1377.
- [JJLM91] C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore. *mural: A Formal Development Support System*. Springer-Verlag, 1991.
- [Jon90] C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, second edition, 1990.
- [Jon03] Cliff B. Jones. The early search for tractable ways of reasoning about programs. *IEEE, Annals of the History of Computing*, 25(2):26–49, 2003.
- [MS98] Erica Melis and Axel Schairer. Similarities and Reuse of Proofs in Formal Software Verification. In Barry Smyth and Pádraig Cunningham, editors, *Proceedings of the 4th European Workshop on Advances in Case-Based Reasoning (EWCBR-98)*, volume 1488 of *LNAI*, pages 76–87, Berlin, September 23–25 1998. Springer.
- [WLB09] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal Methods: Practice and Experience. *ACM Computing Surveys*, 41(4), Oct 2009.