Productive use of failure in top-down formal methods

Alan Bundy
[⋆]School of Informatics
University of Edinburgh
bundy@staffmail.ed.ac.uk

Gudmund Grov
[†]School of Informatics
University of Edinburgh
ggrov@staffmail.ed.ac.uk

Yuhui Lin
[‡]School of Informatics
University of Edinburgh
Y.H.Lin-2@sms.ed.ac.uk

Abstract

Proof obligations (POs) arise when applying formal methods (FMs). The undischarged POs can become a bottleneck in the use of FMs in industry. The work we proposed here, as a part of AI4FM project, aims at increasing the proportion of discharged POs by analysing failed proofs and related patches to classify POs as families and construct proof strategies, which can be used to guide proof search for the POs in the same families.

1 Introduction

Commercial use of formal methods (FM) has had great success in many safety-critical domains such as railway and aviation. There is also increased use in other sectors, for example Microsoft has used FMs to verify device drivers. A recent paper by (Woodcock et al., 2009) provides an up-to-date analysis of a significant number of industrial applications of FMs.

In a top-down approach, FMs are applied early in the development to capture the requirements. The specification is stepwise refined into the final product, typically a piece of software. Many such methods are posit and prove, where a designer posits a step of development and then seeks to justify it. This justification is by proof of generated proof obligations. Examples of such top-down FMs are VDM, B, Event-B and some use of Z.

2 Our approach

As the use of FMs has spread beyond small groups of experts out to far larger groups of industrial engineers, proof automation has become a bottleneck. Although a large proportion of the POs can be discharged by automatic theorem provers, $5 - 10\%$ still requires user interaction, which often requires (expensive) theorem proving experts. For commercial application this can be thousands of proof obligations (POs), requiring many man months or years of work.

We observe that the POs from FMs tend to 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.To illustrate, an industrial case-study using Event-B generated 300 POs, 100 of these required user-interaction, however they could be handled by 5 strategies.

Therefore, we hope to explore this notion of proof families. In particular, we believe that the notion of failures by automatic theorem provers and heuristics can be explored to guide the proof further. Our hypothesis is:

> *we believe that it is possible to classify families of failed proofs within FMs by analysing the context of failures. In each family there exist patterns of proof strategies which can be used to guide the proof search of the blocked proofs.*

Our work is inspired by previous work in *rippling* (Bundy et al., 2005), which is a rewriting technique which works when the goal is embedded in one of the given. Rippling then forces rewriting towards the given, thus guaranteeing termination. It was originally developed for inductive proofs, but has been successfully applied to many other domains. *Proof critics* have been developed for rippling, where the nature of the failure is explored to propose a patch. For example, certain failures suggest a generalisation, others suggests a missing lemma or a different induction rule. Rippling is applicable to certain types of proof obligations, those where a system invariant has to be preserved over the operations. In Grov et al. (2010) we report on a small experiment where use rippling for such POs arising from an Event-B model. A few observations can be made there to illustrate our hypothesis:

- each rippling step has to reduce some measure (to ensure termination) and to preserve what is known as the skeleton, which is the given we are rippling towards. Some steps were not measure reducing while preserving the skeleton. One patch which would have solved this was to introduce a secondary measure on the symbols. In this case the function updates operator was rewritten to set union.
- most rewrite rules were conditional, hence when rippling terminated; the conditions (which were not rippling goals) had to be discharged. Separate patches had to be developed for these.

Our approach will be empirical and iterative, and will include analysis of proof failures and related patches to them. Ideally we would like to use existing (preferable industrial) examples. However, these are the finished articles, which makes it hard to see the productive use of failure that the expert user (implicitly) used to discharge it. Thus we need to see the whole search space including places were failed proof attempts were patched, so to explore these search spaces ourselves.

Through these proofs, we are able to classify POs as families, and extract proof strategies from successful proof attempts. These proof strategies can be used to guide proof search for POs, which are not discharged, in the same families. We will most likely implement and evaluate our patches in Isaplanner (Dixon and Fleuriot, 2003), which is an Isabelle based proof planner, implementing rippling among other methods.

3 Conclusion

We have outlined an approach where we plan to explore the type failures of proofs in top-down formal methods to discover new patches which can be applied to family of proofs. This work is part to the AI4FM project, where the overall goal is to learn high-level proof strategies (and patches from failures) from exemplar proofs provided by expert users. This work is a first step of this project, since we need to understand the typical strategies and patches before we can use machine learning techniques to automatically derive them.

Acknowledgements

This work is supported by EPSRC grant EP/H024204/1: *AI4FM: using AI to aid automation of proof search in Formal Methods*. For more details see `http://www.ai4fm.org`.

References

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.

Lucas Dixon and Jaques Fleuriot. IsaPlanner: A Prototype Proof Planner in Isabelle. In *Proceedings of CADE'03*, volume 2741 of *LNCS*, pages 279–283, 2003.

Gudmund Grov, Alan Bundy, and Lucas Dixon. A small experiement in event-b rippling. In proceedings of AVoCS 2010 and Rodin User and Developer Workshop 2010, April 2010.

J. Woodcock, P.G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)*, 41(4):1–36, 2009.