COMPUTING SCIENCE

Inferring the Proof Process

Andrius Velykis

TECHNICAL REPORT SERIES

No. CS-TR-1344 July 2012

TECHNICAL REPORT SERIES

No. CS-TR-1344 July, 2012

Inferring the Proof Process

A. Velykis

Abstract

This PhD project aims to investigate how enough information can be collected from an interactive formal proof to capture an expert's ideas as a high-level proof process. It would then serve for extracting proof strategies to facilitate proof automation. Ways of inferring this proof process automatically are explored; and a family of tools is developed to capture the different proof processes and their features.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

VELYKIS, A.

Inferring the Proof Process
[By] A. Velykis

Newcastle upon Tyne: Newcastle University: Computing Science, 2012.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1344)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1344

Abstract

This PhD project aims to investigate how enough information can be collected from an interactive formal proof to capture an expert's ideas as a high-level proof process. It would then serve for extracting proof strategies to facilitate proof automation. Ways of inferring this proof process automatically are explored; and a family of tools is developed to capture the different proof processes and their features.

About the authors

Andrius Velykis is a PhD student in AI4FM project at Newcastle University, working under supervision of Prof. Cliff Jones. He is interested in understanding and learning from interactive proof. His PhD research investigates how to extract high-level proof ideas from an interactive proof - these ideas could then be reused to complete similar proofs automatically.

Andrius gained his BSc in Applied Mathematics at Kaunas University of Technology, Lithuania. He continued his studies with MSc Software Engineering at the University of York, UK, where he was awarded the degree with distinction.

Suggested keywords

PROOF PROCESS
INTERACTIVE PROOF
FORMAL METHODS

Inferring the Proof Process

Andrius Velykis

School of Computing Science, Newcastle University, UK
andrius.velykis@newcastle.ac.uk

Abstract. This PhD project aims to investigate how enough information can be collected from an interactive formal proof to capture an expert's ideas as a high-level proof process. It would then serve for extracting proof strategies to facilitate proof automation. Ways of inferring this proof process automatically are explored; and a family of tools is developed to capture the different proof processes and their features.

1 Introduction

Formal methods such as VDM [17], Event-B [2] or Z [27] provide notations and techniques to describe and develop formal specifications and designs. Development steps are justified by discharging relevant (often automatically generated) *proof obligations* (POs). Proofs can be used to establish that the model is consistent (domain/consistency checks), that a detailed design satisfies the properties of abstract specification (data reification proofs), and other properties.

The AI4FM project[1] aims to help users discharge such POs in formal developments on an industrial scale. It investigates how to learn from one interactive proof so that other similar proofs can be completed automatically. Being part of the AI4FM project, this PhD research is dealing with the first steps of learning: capturing, analysing and understanding the expert's proof process.

Current-generation theorem provers include powerful automation techniques and can discharge a large proportion of arising POs. In industrial-size formal developments, however, even a small percentage of remaining POs can amount to a disincentive to deploy formal methods.[2] The general heuristics of automatic theorem provers can fail when faced with complex data structures and proofs that require domain knowledge. In such cases developers revert to interactive proof. The nature of industrial-style proofs (see Sect. 2) allows grouping them into "families" of similar POs. In each family a single proof idea is usually necessary and all other POs in the family are discharged "in a similar way". These are not general proof heuristics—they are specific to the context.

Thus we state the overall hypothesis of the AI4FM research project:

Hypothesis 0. *We believe that it is possible to extract strategies from successful proofs that will facilitate automatic proofs of related POs.*

We approach this problem as a three-step process: (1) *collecting* information about an expert's proof; (2) *extracting* proof strategies; (3) *replaying* the proof

[1] http://www.ai4fm.org

[2] *E.g.* remaining 8% amounted to 2250 POs and over 7 man-months of proof in [1].

strategies to discharge related POs. Current automatic replay techniques, such as *proof planning*, require laborious manual development of reusable proof patterns (see Sect. 2). AI4FM aims to fill this gap by capturing and extracting such patterns (proof strategies) automatically.

The scope of this PhD project is narrower and focuses on the first part of the problem—*collecting* information about the proof—with such hypothesis:

Hypothesis 1. *Enough information can be collected from interactive proof to facilitate understanding of expert's high-level reasoning as reusable proof strategies.*

Some of this information may need to be provided by the expert during the interactive proof. Besides that, the capture should be automatic:

Hypothesis 2. *Certain information about the proof process can be inferred automatically, via analysis of proof context and previous proofs.*

This PhD research aims to provide a generic framework and tools to capture, analyse and infer expert's proof process. The information captured would facilitate extraction of reusable proof strategies. It could also be used for proof visualisation, maintenance or teaching and training (*e.g.* learning by example).

Note that we do not intend to develop a new theorem prover. The aim is to capture and infer high-level proof information. When replaying, we are looking into driving the theorem provers using the high-level strategies instead of constructing the mathematical proofs, thus not affecting their correctness.

2 Background and Related Work

Extracting proof strategies from existing proof and using them to discharge POs would significantly improve automation of formal proofs. This section outlines work in this area and how it relates to this PhD and the AI4FM research project. We also present features of industrial-style proofs, which we expect to be more amenable to proof strategy capture, extraction and reuse.

Proof Planning. Proof planning [5] is a framework to encode a common structure of a family of proofs as high-level proof strategies (methods). A proof plan can be used to guide the proofs of similar theorems. *Proof critics* [14] can provide common patches to help with recovery from failed applications of the associated methods. *Rippling* [6] is a powerful proof planning method, very successful in inductive proofs. Furthermore, tool support is available for the various proof planning techniques, namely IsaPlanner [9] for Isabelle [21].

The *proof methods* (strategies) used by proof planning techniques need to be encoded manually. Some methods allow specifying parts of the strategy (schemas) in a modular way and then dynamically arranging them to find the proof plan [13]. Automatic extraction of strategies from existing proofs is a gap in the process, which AI4FM project aims to address.

Data-Mining Proofs. Some advances have been made in extracting proof strategies by data-mining existing proofs [15,10]. Common patterns of low-level proof steps (*e.g.* rewrite rule applications) can be found by data-mining well-chosen sets of examples. This approach, however, faces difficulty in capturing when the extracted tactics need to be applied. Also, found patterns of low-level proof steps tend to be short and thus of limited reusability. Finally, the need for a corpus of available proofs to learn from would be too restrictive during interactive proof. From the AI4FM perspective, we expect our system to start extracting strategies and assisting the user with similar proofs as soon as one of them is completed.

Proof Reuse & Term Abstraction. Proof reuse after model change has been explored in [22,19]. If model change is small and well-constrained, parts of previous proofs can be reused directly or adapted in a simple way. Proof reuse applicability is identified by comparing shapes of terms in proof goals and hypotheses. Low level details of similar terms can be abstracted using *generalisation* [11,16], *anti-unification* [18] or *schemes* [20].

Industrial POs. Proof obligations generated during industrial use of formal methods (we will call them *industrial-style* POs) exhibit certain features and patterns rarely found in mathematical proofs. The involved formal models consist of complex data structures and operations, featuring a large number of variables and invariants. The POs follow certain method-specific patterns about model consistency, properties or data reification. In addition to that, the core data structures are usually the same (*e.g.* two operations differ in a couple invariants and share large portions of parent data structure), thus large parts of the proofs are very similar. The difficulty of industrial style proofs lies in the size and complexity of involved structures. Their proof strategies exhibit domain-specific features and often require limiting the scope of prover tactic applications. In contrast, mathematical problems and proofs are usually deeper and smaller.

3 Capture and Analyse: *ProofProcess* Framework

The process of developing an interactive proof is rarely straightforward and involves proof exploration, backtracking and "eureka" moments. The final proof is customarily cleaned up and optimised thus disposing of the insights and proof steps that have lead to discovering it. We consider the ideas from the original proof process to be more amenable for reuse in similar proofs and thus more useful than the streamlined final proof. Capturing and analysing them therefore becomes a crucial task, undertaken in the first part of this PhD project.

A number of case studies for proof process analysis were selected initially. These involved new proofs (*e.g.* formal verification of the *Tokeneer ID Station* project [8]), revisiting old proofs by the members of the AI4FM project [12,7,25] as well as doing these proofs using different theorem provers. The exercises focused at discovery of various proof process patterns and similarities as well as identification of waypoints that guide expert's decisions.

The case studies served for the development of a new *ProofProcess* framework to represent, capture, store and analyse all information regarding the proof process. The framework is a core part of this PhD project and consists of two parts: a generic data model of a proof process (abstract model and implementation) and a capture/analysis system.

This section provides an overview of the main ideas on representing and capturing the proof process, as well as general architecture of the framework. The core capture functionality of the *ProofProcess* framework is already available. It establishes the groundwork for development and evaluation of techniques to understand the captured proof process (Sect. 4).

3.1 Model

The *ProofProcess* model is designed to represent a high-level proof process, which is backed by actual proof steps in the theorem prover (*e.g.* command/tactic applications). It provides a flexible data structure, which could accommodate any proof process and any proof structure that the expert wishes to describe the proof attempt by.[3] The low-level proof steps are augmented with additional "why" information about the expert's proof process:

- Proof *granularity*: we want different layers of abstraction: from high-level proof plan to the tactic steps. The proof needs to be captured at any arbitrary granularities that the expert deems appropriate.
- Proof *structure*: while the proofs are usually done as a sequence of steps, the actual structure is rarely linear (*e.g.* the base and step cases of inductive proofs can be seen as branches in proof trees).
- Multiple proof *attempts*: we want to capture the whole proof development. Failed attempts may still contain generally applicable proof steps, even though they were not successful in that particular case. All versions leading to a finished proof should be captured.
- *Intent*: high-level explanation of expert's proof direction. These are simple tags that give names and description to underlying proof reasoning (*e.g.* akin to *Isolation*, *Collection* and *Attraction* in [4, Ch. 12]). The *intent* is coupled with *features* (below) to paint the whole proof process picture.
- Proof *features*: anything that drives the expert's proof. It can be the shape of the goal, the available lemma, certain datatypes or records, *etc.* Such information is usually readily available during the proof process, however deducing it from the finished proof is difficult. The features are used to pinpoint what triggered a proof step, change in proof structure or direction. By adapting or generalising them, we would know when the strategies extracted from this proof attempt apply in the next one.

A generic model allows building reusable analysis techniques on top of it. The desire is that adequate proof *intent* and *feature* combinations would be provided or

[3] It is being tested with different theorem provers and different proof structures, *e.g.* "chain" of command applications in Z/EVES [23] or a structured *Isar* proof [26].

inferred during proof capture. Then generic analysis techniques and tools could work without the knowledge of prover-specific terms or commands. If needed, prover-specific model extensions allow representing detailed proof context.

The abstract *ProofProcess* model is developed as a formal Z specification. The *ProofProcess* system (Sect. 3.2) has a concrete implementation in EMF/Java.

3.2 System

The industrial-style proofs are usually of significant size and capturing the proof process manually immediately becomes too cumbersome. For this reason, a *ProofProcess* system is being built alongside the model to develop and test techniques for capture and analysis of the proof process.

The tools have extensible and modular architecture. They are built on modern platforms of Java, Eclipse, and EMF [24]. The system works as an add-on to theorem prover assistants:[4] user is not forced into new habits of using the prover. Instead, the tool "wire-taps" the link with the theorem prover to capture basic proof information and performs analysis calculations in parallel. A close integration is necessary to avoid losing the proof information.

The captured proof activities are subjected to analysis in order to try to infer certain information about the proof process automatically (as proposed in Hyp. 2). The design allows for modular "matcher" routines, which would calculate the proof process features, structure or intent. These would then be combined to determine the most applicable solution. One approach of combining different "matchers" using weights is described in [13].

Initial "matchers" already available in the *ProofProcess* system can recognise proof re-runs, diverging proof attempts, or basic proof structure, *e.g.* parallel case splits. Plans for further proof process analysis routines are discussed next.

4 Understand and Learn: Next Steps

This section presents further ideas on analysing and understanding the captured proof process. These are directions that will be explored in the remainder of this PhD research. By building upon the core functionality of the *ProofProcess* framework, we focus on capturing features of industrial-style proofs and inferring the proof process automatically by learning from previous examples.

4.1 Role of Lemmas

Existing proof reuse/proof planning techniques focus on the shapes of goal terms, *e.g.* a term is inductive or the goal has a certain *skeleton* [6,13,19]. We believe that lemma usage (and extraction) is an important proof feature, especially in industrial-style proofs with similar data structures. Intermediate lemmas about

[4] Implementations for Isabelle/HOL [21] and Z/EVES [23] are available; integration with Rodin Tools [3] is in plans.

data structures significantly facilitate automatic proof, however the lemmas are not obvious and are closely related to the specific structure and characteristics of the model. The need for such a lemma would be discovered by the expert during an interactive proof attempt. Proof *features* could be used to describe the need for the lemma and its properties, *e.g.* how it is related to the data structure. Then a need for a similar lemma would be recognised when a similar PO is encountered next time. Given enough information about the original lemma, the new lemma can be generated or adapted, and the general proof strategy reused.

Capturing used lemmas is paramount for highly automatic proof tactics, *e.g.* ones that perform multiple goal rewrites and transformations.[5] The fact that a specific lemma was required is not obvious from the tactic application and naive reuse would easily fail, leaving the user puzzled. Furthermore, certain lemmas (*e.g.* associativity, equality substitution) would help recognise specific proof patterns and understand the proof process better.

4.2 Affected Goal Terms

Industrial scale POs can total hundreds of terms in the proof goal arising from complex data structures. This prevents use of automated tactics and requires careful interactive guidance (*i.e.* to make sure that only interesting/required hypotheses are used). By analysing the goals before and after tactic application we could try to identify affected terms. This would provide several advantages:

- *Understanding the proof process*: by following the narrow path of affected goal terms, we could more easily understand the expert's path of reasoning.
- *Reuse information about "unimportant" terms*: by inverting the scope, we may isolate the terms which never change in certain proofs, thus limiting proof search in similar POs.
- *Extracting a* toy *problem*: the affected terms would constitute a minimal example representing the given problem. It could be extracted as a simplified view of the PO, and analysed without the burden of surplus hypotheses.

A similar approach was taken in [19], where hundreds of hypotheses in POs were filtered to analyse which of them were used in the proof. The knowledge was used to check whether a proof reuse was possible (*e.g.* if only unused hypotheses have changed). *Proof planning* techniques do not perform such analysis, because they are usually applied to much smaller examples.

The approach can be refined further by investigating affected fields in complex data structures; or important features of complex functions. For example, when faced with a list reverse function rev in the proof, the important proof feature can be that rev is defined in terms of list append function append.

4.3 "Fuzzy" Matching

The *ProofProcess* model is designed to be flexible and allow the expert to specify any proof process with arbitrary proof features. One of the aims of this PhD

[5] *E.g.* auto in Isabelle/HOL [21], prove by reduce in Z/EVES [23].

project is to reuse this information automatically when a similar PO is encountered. Recognising the similarities with a previously captured proof process is the difficult part of achieving this objective. By performing generalisation and partial matching of properties and intents with some uncertainty—we call this "fuzzy" matching—we aim to recognise the proof process being undertaken. The *ProofProcess* model allows setting proof features and intent on every level of the proof tree. When analysing the current proof, we can match it with previous proofs or their subproofs. The ultimate objective is to recognise that something similar has been attempted before, and infer that the proof process is similar. Note that similar approach and techniques would be applied when extracting the proof strategies from the collected proof process information. The difference is that when inferring the proof process, we have access to more data at any proof step—namely the application of the proof step tactic, and a resulting goal—than when guiding the proof search with an extracted proof strategy.

A step even further would be analysing custom proof features. We anticipate some of the proof properties to be quite abstract, domain specific, or simply difficult to express formally. For example, an expert may wish to indicate that a function is "addable". Such property is difficult to express formally, but if encountered during interactive proof, the expert may easily relate it to appropriate proof strategy. The difficulty lies with matching such proof properties in similar proofs automatically, because we lack semantics for this property. One approach may be to check what other properties can be inferred when this property is specified, and relate them in some way.

The *ProofProcess* model is designed to be generic, thus custom properties are supported. There is a possibility to provide an extensible mechanism (likely prover-specific), which could allow specifying semantics for the custom properties. The properties could be linked to custom "matchers", which would be used in the general framework.

Acknowledgements. Thanks to Leo Freitas, Gudmund Grov, Cliff Jones and other AI4FM members for the useful input to the *ProofProcess* framework. This PhD research is supported by EPSRC grant EP/H024050/1 (AI4FM).

References

1. Abrial, J.R.: Formal methods: Theory becoming practice. J. UCS 13(5), 619–628 (2007)
2. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
3. Abrial, J.R., Butler, M.J., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. STTT 12(6), 447–466 (2010)
4. Bundy, A.: The Computer Modelling of Mathematical Reasoning. Academic Press Professional, Inc., San Diego, CA, USA, digital (2010) edn. (1985)
5. Bundy, A.: The use of explicit plans to guide inductive proofs. In: Lusk, E., Overbeek, R. (eds.) CADE. LNCS, vol. 310, pp. 111–120. Springer (1988)

6. Bundy, A., Basin, D., Hutter, D., Ireland, A.: Rippling: Meta-level Guidance for Mathematical Reasoning, Cambridge Tracts in Theoretical Computer Science, vol. 56. Cambridge University Press (June 2005)
7. Butterfield, A., Freitas, L., Woodcock, J.: Mechanising a formal model of flash memory. Sci. Comput. Program. 74(4), 219–237 (2009)
8. Copper, D., Barnes, J.: Tokeneer ID station EAL5 demonstrator: Summary report. Tech. Rep. S.P1229.81.1 Issue: 1.1, Altran-Praxis (August 2008)
9. Dixon, L., Fleuriot, J.: IsaPlanner: A prototype proof planner in Isabelle. In: Baader, F. (ed.) CADE, LNCS, vol. 2741, pp. 279–283. Springer (2003)
10. Duncan, H.: The Use of Data-Mining for the Automatic Formation of Tactics. Ph.D. thesis, School of Informatics, University of Edinburgh (2007)
11. Felty, A.P., Howe, D.J.: Generalization and reuse of tactic proofs. In: Pfenning, F. (ed.) LPAR. LNCS, vol. 822, pp. 1–15. Springer (1994)
12. Freitas, L., Woodcock, J.: Mechanising Mondex with Z/Eves. Formal Asp. Comput. 20(1), 117–139 (2008)
13. Heneveld, A.: Using Features for Automated Problem Solving. Ph.D. thesis, School of Informatics, University of Edinburgh (February 2006)
14. Ireland, A., Bundy, A.: Productive use of failure in inductive proof. J. Autom. Reasoning 16(1-2), 79–111 (1996)
15. Jamnik, M., Kerber, M., Pollet, M., Benzmüller, C.: Automatic learning of proof methods in proof planning. Logic Journal of the IGPL 11(6), 647–673 (2003)
16. Johnsen, E.B., Lüth, C.: Theorem reuse by proof term transformation. In: Slind, K., Bunker, A., Gopalakrishnan, G. (eds.) TPHOLs. LNCS, vol. 3223, pp. 152–167. Springer (2004)
17. Jones, C.B.: Systematic Software Development using VDM. Prentice Hall International, second edn. (1990)
18. Krumnack, U., Schwering, A., Gust, H., Kühnberger, K.U.: Restricted higher-order anti-unification for analogy making. In: Orgun, M.A., Thornton, J. (eds.) Australian Conference on Artificial Intelligence. LNCS, vol. 4830, pp. 273–282. Springer (2007)
19. Mehta, F.: Supporting proof in a reactive development environment. In: SEFM. pp. 103–112. IEEE Computer Society, London, England, UK (2007)
20. Montano-Rivas, O., McCasland, R.L., Dixon, L., Bundy, A.: Scheme-based synthesis of inductive theories. In: Sidorov, G., Aguirre, A.H., García, C.A.R. (eds.) MICAI (1). LNCS, vol. 6437, pp. 348–361. Springer (2010)
21. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (May 2002)
22. Reif, W., Stenzel, K.: Reuse of proofs in software verification. In: Shyamasundar, R. (ed.) Foundations of Software Technology and Theoretical Computer Science, LNCS, vol. 761, pp. 284–293. Springer (1993)
23. Saaltink, M.: The Z/EVES system. In: Bowen, J., Hinchey, M., Till, D. (eds.) ZUM '97. LNCS, vol. 1212, pp. 72–85. Springer (1997)
24. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional, second edn. (December 2008)
25. Velykis, A., Freitas, L.: Formal modelling of separation kernel components. In: Cavalcanti, A., Déharbe, D., Gaudel, M.C., Woodcock, J. (eds.) ICTAC. LNCS, vol. 6255, pp. 230–244. Springer (2010)
26. Wenzel, M.M.: Isabelle/Isar - a versatile environment for human-readable formal proof documents. Ph.D. thesis, Technische Universität München (2002)
27. Woodcock, J., Davies, J.: Using Z: Specification, Refinement, and Proof. Prentice Hall International (March 1996)