Scaling Automated Theory Exploration

Chris Warburton and Ekaterina Komendantskaya

University of Dundee,
http://tocai.computing.dundee.ac.uk

Abstract. We investigate the **theory exploration** (TE) paradigm for computer-assisted Mathematics and identify limitations and improvements for current approaches. Unlike the theorem-proving paradigm, which requires user-provided conjectures, TE performs an open-ended search for theorems satisfying given criteria. We see promise in TE for identifying new abstractions and connections in libraries of software and proofs, but realising this potential requires more scalable algorithms than presently used.

1 Introduction

Given a signature Σ and a set of variables V, we call the pair (Σ, V) a *theory* and use *theory exploration* (TE) to refer to any process $(\Sigma, V) \overset{TE}{\to} \text{Terms}(\Sigma, V)$ for producing terms of the theory which are well-formed, provable and satisfy some criterion referred to as "interesting". These conditions give rise to the following questions, which we use to characterise TE systems:

Q1 How do we generate terms?
Q2 How do we guarantee well-formedness?
Q3 How do we prove terms?
Q4 What is considered "interesting"?

Early implementations like Theorema [2] provided interactive environments, similar to computer algebra systems and interactive theorem provers, to assist the user in finding theorems. In this setting, terms are formed by the user in whichever way they find interesting, whilst the software provides support for **Q2** and **Q3**.

Subsequent systems have investigated *automated* theory exploration, for tasks such as lemma discovery [7]. By removing user interaction, **Q1** and **Q4** must be solved by algorithms. In existing systems these are tightly coupled to improve efficiency, which makes it difficult to try different approaches independently.

As an example, QuickSpec [4] discovers equations about Haskell code, which are defined as "interesting" if they cannot be simplified using previously discovered equations. The intuition for such criteria is to avoid special cases of known theorems, such as $0 + 0 = 0$, $0 + 1 = 1$, etc. when we already know $0 + x = x$. Whilst **Q4** is elegantly implemented with a congruence closure relation (version 1) and a term rewriting system (version 2), the term generation for **Q1** is performed by brute-force.

Although QuickSpec only *tests* its equations rather than proving them, it is still used as the exploration component of more rigorous systems like HipSpec and Hipster.

In the following, we give an overview of the state of the art in automated theory exploration, then present potential improvements and our initial attempts at implementation.

2 Theory Exploration in Haskell

Automated theory exploration has been applied to libraries in Isabelle and Haskell, although we focus on the latter as its implementations are the most mature (demonstrated by the fact that Hipster explores Isabelle by first translating it to Haskell). Haskell is interesting to target, since its use of pure functions and algebraic datatypes causes many programs to follow algebraic laws. However, since Haskell's type system cannot easily encode such laws, less effort is given to finding and stating them; compared to full theorem provers like Isabelle. Hence we imagine even a shallow exploration of code repositories such as Hackage could find many interesting theorems.

Currently, the most powerful TE system for Haskell is HipSpec, which uses off-the-shelf automated theorem provers (ATPs) to verify the conjectures of QuickSpec. QuickSpec, in turn, enumerates all type-correct combinations of the terms in the theory up to some depth, groups them into equivalence classes using the QuickCheck counterexample finder, then conjectures equations relating the members of these classes. This approach works well as a lemma generation system, making HipSpec a capable inductive theorem prover as well as a theory exploration system [3].

3 The ML4HS Framework

We consider **Q2** and **Q3** to be adequately solved by the existing use of type systems and ATPs, respectively. We identify the following potential improvements for the other questions:

Q1 Enumerating all type-correct terms is a brute-force solution to this question. Scalable alternatives to brute-force algorithms are a well-studied area of Artificial Intelligence and Machine Learning. In particular, heuristic search algorithms like those surveyed in [1] could be used. We could also use Machine Learning methods to identify some sub-set of a given theory, to prioritise over the rest.

Q4 Various alternative "interestingness" criteria have been proposed, for example those surveyed in [5]. Augmenting or replacing the criteria may be useful, for example to distinguish useful relationships from incidental coincidences; or to prevent surprising, insightful equations from being discarded because they can be simplified.

We are implementing a system called ML4HS to investigate these ideas. Its current form is a pre-processor for QuickSpec for prioritising theory elements. Inspired by the use of premise selection [9] to reduce the search space in ATP, we select sub-sets of the given theory to explore, chosen to try and keep together those expressions which combine in interesting ways, and to separate those which combine in uninteresting ways.

We hypothesise that similarity-based clustering of expressions, inspired by that of ML4PG [8] and related work in ACL2 [6], is an effective method for performing this separation. Future experiments will test this by comparing the throughput of QuickSpec with and without the ML4HS pre-processor.

Acknowledgements

Thank you to the HipSpec team at Chalmers University (Moa Johansson, Koen Claessen, Nick Smallbone, Dan Rosén and Irene Lobo Valbuena) for useful discussions of these ideas.

References

1. Christian Blum, Jakob Puchinger, Günther R Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.
2. Bruno Buchberger. Theory exploration with theorema. *Analele Universitatii Din Timisoara, ser. Matematica-Informatica*, 38(2):9–32, 2000.
3. Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating inductive proofs using theory exploration. In *Automated Deduction–CADE-24*, pages 392–406. Springer, 2013.
4. Koen Claessen, Nicholas Smallbone, and John Hughes. Quickspec: Guessing formal specifications using testing. In Gordon Fraser and Angelo Gargantini, editors, *Tests and Proofs*, volume 6143 of *Lecture Notes in Computer Science*, pages 6–21. Springer Berlin Heidelberg, 2010.
5. Liqiang Geng and Howard J Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.
6. Jónathan Heras, Ekaterina Komendantskaya, Moa Johansson, and Ewen Maclean. Proof-pattern recognition and lemma discovery in acl2. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 389–406. Springer, 2013.
7. Moa Johansson, Dan Rosn, Nicholas Smallbone, and Koen Claessen. Hipster: Integrating theory exploration in a proof assistant. In StephenM. Watt, JamesH. Davenport, AlanP. Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics*, volume 8543 of *Lecture Notes in Computer Science*, pages 108–122. Springer International Publishing, 2014.
8. Ekaterina Komendantskaya, Jónathan Heras, and Gudmund Grov. Machine Learning in Proof General: Interfacing Interfaces. In Cezary Kaliszyk and Christoph Lüth, editors, *UITP*, volume 118 of *EPTCS*, pages 15–41, 2013.
9. Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In *Automated Reasoning*, pages 378–392. Springer, 2012.