Towards Big Theory Exploration

Sólrún Halla Einarsdóttir[1] and Moa Johansson[1]

Chalmers University of Technology, Gothenburg, Sweden.
{slrn, moa.johansson}@chalmers.se

Abstract

QuickSpec is a system for theory exploration, able to automatically generate many interesting conjectures about mathematical functions. However, when exploring bigger theories containing many different functions (approx. $> 15 \sim 20$), the exponential blow-up of its search space can make it too slow to be useful. We present work in progress on a template-based extension, intended as a complement when dealing with big theories. We sacrifice broad search for more direction towards commonly occurring property patterns, sourced from e.g. mathematical libraries.

1 Introduction

Theory exploration is a method of automatically inventing interesting properties or candidate lemmas. For example, we might give our theory exploration system the functions *length* and *reverse* on lists, and discover the property $length\ (reverse\ xs) = length\ xs$. Knowing about this property could then be helpful in inventing or proving more complicated theorems.

QuickSpec [6] is a theory exploration tool which discovers equational properties about Haskell programs by generating all type-correct terms that can be formed using the given functions, up to a size limit, and then using the property-based testing tool QuickCheck [1] to test which terms are equivalent. A weakness of QuickSpec is that if it is given a large number of functions to explore at once it will cease to be quick as its name suggests and instead becomes too slow to be practically useful, while outputting an overwhelming amount of properties and struggling to prove away those that are uninteresting (see example in section 4.2 of [6]).

Determining what properties are "interesting" is a big challenge. We hypothesize that many properties that humans consider interesting (or useful) in fact often have similar shapes (some of these shapes have names, such as associativity, commutativity, distributivity, or appear as type-class laws etc.). Also, if considering using the theory exploration system in combination with a theorem prover, as with QuickSpec in the Hipster [4] system, a failed proof attempt might suggest shapes of potential missing lemmas [3]. We therefore propose a "quick-QuickSpec" using *property templates* to capture such shapes or patterns, while sacrificing some of the completeness in search. The templates are instantiated with available function symbols and results tested for counter-examples. Surviving conjectures are presented to the user or passed on to an automated prover. Similar techniques have been suggested in e.g. [5, 2], but we aim for a higher level of automation.

2 Template-based QuickSpec

We have implemented a prototype of a modified version of QuickSpec using templates. The user specifies each template they are interested in using an expression format where question marks denote holes, for example: $?F(?G(X,Y)) = ?F(?G(Y,X))$ describes the composition of two functions being commutative in two variables. Candidate properties are generated by

attempting to fill the holes in the template using the functions in the exploration scope, restricting the generated equations to be well typed. For example, filling the holes in the template above using functions *length*, *reverse*, and ++ on lists gives the candidate properties $length\,(xs\,+\!+\,ys) = length\,(ys\,+\!+\,xs)$ (cp1) and $reverse\,(xs\,+\!+\,ys) = reverse\,(ys\,+\!+\,xs)$ (cp2). The generated candidate properties are then tested using QuickCheck [1]. If no counterexamples are found the property is presented to the user as a conjecture. In our example, cp1 passes this phase and is presented as a conjecture, while counterexamples are found to cp2.

Evaluation

We have compared the performance of our extension to standard QuickSpec on some benchmark theories[1]. Using 12 templates describing basic properties of functions and operators, we first explored a few normal-sized theories of list functions, booleans and arithmetic. Most of the properties found by standard QuickSpec for these theories are in fact also found using these templates, and some of the properties not replicated are ones we consider redundant or uninteresting. We even find some nice properties that standard QuickSpec had pruned away (e.g. one of De Morgan's laws).

Secondly, we considered a stress-test (section 4.2 in [6]), where QuickSpec was used to find properties about a set of 33 Haskell functions on lists. This took standard QuickSpec 42 minutes and resulted in 398 properties when limited to terms of size 7 or less, and hit a time limit of 2 hours when the size was increased to 8, illustrating how running QuickSpec on larger theories scales poorly with regard to run-time and may produce an overwhelming amount of output. In contrast, running our new prototype on this big theory, using the same standard 12 templates as above, we discover 41 properties in under 2 seconds, including some properties containing terms of size 7. We can also discover properties containing even larger terms (e.g. terms of size 9) in under 1 second if we provide templates supporting those sizes. Theory exploration is now tractable, at the price of providing a stricter specification of the shape of desired properties.

3 Next steps

The next step is to automate the discovery of lemma templates. We will explore several options, e.g. machine learning to extract common patterns from proof libraries, learning common lemma shapes given properties of the theorem we want to prove (c.f. [2]), as well as exploiting type-class laws and other algebraic properties. We will also investigate extracting templates from failed proof attempts, similar to critics in proof planning [3]. We will conduct a larger experimental evaluation, comparing standard QuickSpec with our template-based algorithm to identify the "sweet-spot" for the respective approaches, and how they can be combined. Naturally, the template based approach might miss "unusually shaped" properties, but this could be a price worth paying for scalability. Perhaps the more exhaustive approach of standard QuickSpec should be used for small terms (of which there are fewer) and a template-based approach used for larger terms, with the exact split depending on the size of the theory being explored.

4 Acknowledgements

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation.

[1] See https://github.com/solrun/quickspec/tree/master/template-examples

References

[1] K. Claessen and J. Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. In *Proceedings of ICFP*, pages 268–279, 2000.

[2] J. Heras, E. Komendantskaya, M. Johansson, and E. Maclean. Proof-pattern recognition and lemma discovery in ACL2. In *Proceedings of LPAR*, 2013.

[3] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16:79–111, 1996.

[4] M. Johansson. Automated theory exploration for interactive theorem proving: An introduction to the Hipster system. In *Proceedings of ITP*, volume 10499 of *LNCS*, pages 1–11. Springer, 2017.

[5] O. Montano-Rivas, R. McCasland, L. Dixon, and A. Bundy. Scheme-based theorem discovery and concept invention. *Expert systems with applications*, 39(2):1637–1646, 2012.

[6] N. Smallbone, M. Johansson, K. Claessen, and M. Algehed. Quick specifications for the busy programmer. *Journal of Functional Programming*, 27, 2017.