AUTOMATED CONJECTURING IN QUICKSPEC

Moa Johansson & Nicholas Smallbone
Department of Computer Science and Engineering
Chalmers University of Technology
412 96, Gothenburg, Sweden
{jomoa, nicsma}@chalmers.se

ABSTRACT

A key component of mathematical reasoning is the ability to formulate interesting conjectures about a problem domain at hand. This task has not yet been widely studied by the automated reasoning and AI communities, but we believe interest is growing. In this paper, we give a brief overview of a theory exploration system called QuickSpec, able to automatically discover interesting conjectures about a given set of functions. QuickSpec works by interleaving term generation with random testing to form candidate equational conjectures. This is made tractable by starting from small sizes and ensuring that only terms that are irreducible with respect to already discovered equalities are considered. QuickSpec has been successfully applied to generate lemmas for automated inductive theorem proving as well as to generate specifications of functional programs. We also give a small survey of different approaches to conjecture discovery, and speculate about future directions combining symbolic methods and machine learning.

1 INTRODUCTION

What makes a conjecture interesting and worth trying to prove as a lemma? For a human mathematician the motivation might be that the statement would make another proof shorter, clearer and easier to understand, or that the lemma captures, for instance, some common algebraic property of a structure of interest. Coming up with the right lemma in the right situation is sometimes described as a *eureka step*: a sudden insight that makes a solution almost obvious.

However, automating these kind of creative steps in the context of automated reasoning is difficult both for symbolic and data-driven methods: Interesting lemmas might not fall out from simply applying deductive rules to axioms in a theorem prover; and when considering a new mathematical theory or structure, we might not have a lot of previous proofs in the domain, which makes it difficult to directly apply traditional machine learning techniques.

In the context of automation of inductive proofs, there has been work on patching failed proof attempts by speculating lemmas using e.g. various *proof critics*, ranging from simply generalising the goal by replacing common subterms with new variables, to more complex methods requiring specialised heuristics and search (Boyer & Moore, 1979; Ireland & Bundy, 1996; Dixon & Johansson, 2007). However, we will here describe a different approach, *theory exploration*, which instead of trying to construct lemmas from specific proof attempts, proceeds bottom-up: given a set of concepts, what interesting conjectures can we come up with?

The theory exploration tool QuickSpec (Smallbone et al., 2017) generates conjectures about a set of functions given by the user. It works by enumerating terms starting from small sizes up to a user specified limit, and then evaluating them at random values to determine if any terms appear to be equal. To avoid uninteresting conjectures, and manage the search space, only terms that have a unique normal form with respect to what has been discovered so far are kept. To give the reader a flavour of what the system can do, figure 1 shows the *complete* output of QuickSpec on the theory of natural numbers with addition, multiplication and greatest common divisor, implemented as functions in Haskell. QuickSpec takes about 5 seconds to run on this example. Note that QuickSpec has no built-in knowledge of the laws of arithmetic, so everything in figure 1 is invented from scratch. These laws include common properties such as commutativity, associativity, distributivity and iden-

```
 1. x+y = y+x                    12. gcd(x,y) = gcd(y,x)
 2. x+0 = x                      13. gcd(x,x) = x
 3. (x+y)+z=x+(y+z)              14. gcd(x,0) = x
 4. x*y = y*x                    15. gcd(x,1) = 1
 5. x*0 = 0                      16. gcd(x,x*y) = x
 6. x*1 = x                      17. gcd(x,x+y) = gcd(x,y)
 7. (x*y)*z = x*(y*z)            18. gcd(gcd(x,y),z) = gcd(x,gcd(y,z))
 8. x*(y+y) = y*(x+x)            19. gcd(x*y,x*z) = x*gcd(y,z)
 9. x*(y+1) = x+(x*y)            20. gcd(x*x,y*y) = gcd(x,y)*gcd(x,y)
10. (x*y)+(x*z) = x*(y+z)        21. gcd(x*y,z+y) = gcd(x*z,z+y)
11. x*(y+(y+y)) =                22. gcd(x+x,y+y) = gcd(x,y)+gcd(x,y)
    y*(x+(x+x))                  23. gcd(x+y,y+y) = gcd(x+x,x+y))
                                 24. gcd(x*x,1+1) = gcd(x,1+1)
```

Figure 1: Complete QuickSpec output on a small theory about arithmetic.

tity laws, as well as more specific facts about `gcd`, such as law 19 (about numbers having common factors), law 20 (about squares), the curious interchange law 21, and law 24 (which holds because 2 is prime). Notice that after discovering for instance law 5, $x * 0 = 0$, QuickSpec will not generate any terms where $x * 0$ is a subterm, as such a term would be reducible by applying law 5 as a rewrite rule, and therefore is not considered interesting. This is one of the key observations that makes term generation tractable (see Smallbone et al. (2017) for full technical details and optimisations of the term generation algorithm). Note that QuickSpec itself does not prove any properties, but tests them thoroughly on randomly generated values, using Haskell's QuickCheck tool (Claessen & Hughes, 2000). Depending on the theory we are exploring we can of course pass the properties discovered by QuickSpec on to a suitable automated or interactive theorem prover.

QuickSpec was originally designed to automatically discover equational properties about functional programs written in Haskell, i.e. an algebraic specification (hence the name), but has also successfully been used to discover lemmas for automated inductive provers, allowing improvements in automation beyond the previous state-of-the-art (Claessen et al., 2013), and to help construct theories in the interactive proof assistant Isabelle (Johansson, 2017).

2 Two Examples from Mathematics

To demonstrate that QuickSpec can also be applied to non-trivial and unfamiliar mathematical theories, we now use it to explore two *non-associative algebras*: the octonions and Jordan algebras.

The octonions. Octonions (Baez, 2002) are the lesser known of the four normed division algebras, the others being the real numbers, complex numbers and quaternions. They have some exotic properties: unlike the reals and complex numbers, multiplication on octonions is neither associative nor commutative. However, there are many interesting properties that they do satisfy. In Smallbone et al. (2017), we investigated how many such properties QuickSpec could find about octonion multiplication and inverse, and we repeat that experiment here. QuickSpec took less than two seconds to run and produced the 14 conjectures shown in Figure 2 (and no others). How many of these properties are interesting? We make the assumption that human mathematicians give names to interesting properties, and find that of the 14 properties, 12 have standard names[1]. Note that the unnamed properties are consequences of *diassociativity*: expressions with two variables can be reassociated arbitrarily[2].

For this experiment, the octonions were modelled as a type in a Haskell program, using the Cayley-Dickson construction: octonions are expressed as pairs of quaternions, which in turn are represented as pairs of complex numbers, which then are pairs of reals. The reader should note that QuickCheck

[1] See e.g. `https://groupprops.subwiki.org/wiki/`. and `https://en.wikipedia.org/wiki/Quasigroup`

[2] QuickSpec only kept (11) as it happened to discover it before the Moufang identity (13). In Smallbone et al. (2017), the Moufang identity was discovered first because of a different term order, so (11) was pruned away.

$1^{-1} = 1$	inverse of identity element (1)
$x * 1 = x$	right identity axiom (2)
$1 * x = x$	left identity axiom (3)
$(x^{-1})^{-1} = x$	involution of inverse (4)
$x * x^{-1} = 1$	right inverse axiom (5)
$(x * x) * y = x * (x * y)$	left alternative loop property (6)
$(x * y) * x = x * (y * x)$	flexible loop property (7)
$(x * y) * y = x * (y * y)$	right alternative loop property (8)
$x^{-1} * y^{-1} = (y * x)^{-1}$	antiautomorphic inverse property (9)
$x^{-1} * (x * y) = x$	left inverse property (10)
$x * (y * (y * x)) = (x * y) * (y * x)$	(11)
$x * (y * (y * y)) = (x * y) * (y * y)$	(12)
$x * ((y * z) * x) = (x * y) * (z * x)$	Moufang identity (13)
$(x * (y * x)) * z = x * (y * (x * z))$	left Bol loop property (14)

Figure 2: Equalities discovered by QuickSpec about multiplication and inverse on the octonions.

cannot use arbitrary real numbers for testing, as most are uncomputable. Instead, the terms are tested and evaluated on those octonions where each component is a rational number. This introduces the risk that QuickSpec could produce false positives, but in this case, since the octonion operations are built only from continuous functions on the real numbers such as multiplication, any equations that hold for rational octonions in fact hold for all octonions.

Jordan algebras. *Jordan algebras* are non-associative algebras first invented to model measurements in quantum mechanics (Jordan et al., 1934). In a Jordan algebra, multiplication is commutative and satisfies the *Jordan identity*, $(xy)(xx) = x(y(xx))$. As our Jordan algebra we took *Hermitian matrices*[3] equipped with the *Jordan product*, $A \circ B := (AB + BA)/2$. We gave QuickSpec the operators $\circ$ and $+$ and the identity and zero matrices, and a test data generator for random Hermitian matrices. Generating general Hermitian matrices is however non-trivial, due to compatibility issues (as for the octonions). We therefore restrict our generator function to a subset of Hermitian matrices which are computable: symmetric square matrices over the rationals.

In a few seconds, QuickSpec produced equations (1)–(8) of Figure 3. We see that $\circ$ is commutative (1) and satisfies the Jordan identity (7). It is not associative, but it does have the expected zero and identity elements (2, 3), and distributes over $+$ (6). The remaining laws (4, 5, 8) are consequences of distributivity: (4, 5) are kept because they happen to be discovered *before* distributivity, and 8 because QuickSpec was unable to prove it. In all, of the 8 equations, 5 appear to be interesting.

The Jordan identity (7) perhaps looks strange, but is designed to make the algebra *power-associative*: in a Jordan algebra, the expression $x^n = x \circ \ldots \circ x$ has the same value no matter how the expression is bracketed. To see if QuickSpec could discover power-associativity, we added the operator x^n (with an arbitrary bracketing) and re-ran QuickSpec. After a few seconds, it found equations (9)–(16). Most of these are normal properties of the power operation, and suggest that, even though though multiplication is non-associative, powers work as usual. In particular, 15 implies that the algebra is power-associative. Equation 16 suggests the generalisation $A^m \circ (A^n \circ B) = A^n \circ (A^m \circ B)$, an important property of Jordan algebras, which is found if the maximum term size is increased to 9.

Summary. In both examples, we see that QuickSpec can take an unfamiliar structure and find familiar laws about it. These laws help to demystify the structure, and can be a useful start for a mathematician who wants to understand how the structure behaves.

[3]A Hermitian matrix is a square matrix with entries in the complex numbers which is its own conjugate transpose. That is, each entry A_{ij} is equal to the complex conjugate of A_{ji}.

$$A \circ B = B \circ A \quad (1)$$
$$A \circ 0 = 0 \quad (2)$$
$$A \circ I = A \quad (3)$$
$$A \circ (B + B) = B \circ (A + A) \quad (4)$$
$$A \circ (B + I) = A + (A \circ B) \quad (5)$$
$$(A \circ B) + (A \circ C) = A \circ (B + C) \quad (6)$$
$$A \circ (B \circ (A \circ A)) = (A \circ A) \circ (A \circ B) \quad (7)$$
$$A \circ (B + (B + B)) = B \circ (A + (A + A)) \quad (8)$$
$$A^1 = A \quad (9)$$
$$0^n = 0 \quad (10)$$
$$I^n = I \quad (11)$$
$$(A^m)^n = A^{mn} \quad (12)$$
$$(A \circ A)^n = A^{n \circ n} \quad (13)$$
$$A^{n+1} = A \circ A^n \quad (14)$$
$$A^m \circ A^n = A^{m+n} \quad (15)$$
$$A \circ (B \circ A^n) = A^n \circ (A \circ B) \quad (16)$$

Figure 3: Equalities discovered by QuickSpec about Jordan multiplication and power.

The source code for QuickSpec, together with the example code, are available online at `https://github.com/nick8325/quickspec`.

3 Related Work

Conjecture generation systems tend to roughly fall into three categories: heuristic rule-based systems, term generation-and-testing (to which QuickSpec belongs) and neural network-based systems.

Rule-based and Heuristic. The AM system is probably the first system designed to discover mathematical concepts (Lenat, 1976). It relied on several hundred heuristic rules to generate mathematical concepts and conjectures, with each rule contributing a pre-defined "interestingness score" to the result. The HR system had a similar goal of automating concept formation and conjecturing (Colton, 2002). It took a set of axioms as inputs, and combined model finding with heuristic production rules to invent new conjectures of interest. The MATHsAiD system was specifically designed as a theory exploration system for mathematicians (McCasland et al., 2017). The discovery process was guided by a forward reasoning process, which instantiated templates called *theorem shells*, following heuristic plans constructed to reflect human mathematical reasoning.

In comparison, QuickSpec is a more light-weight system concerned only with conjecturing, not any other concept formation tasks. It does not employ explicit heuristic rules: it simply generates terms about the given concepts, and evaluates them on random values to learn which ones appear to be equal. It also uses a very simple notion of interestingness: a term is interesting to explore if it is non-reducible with respect to what is known so far. This makes it more flexible, as it can be used to explore conjectures about any datatype for which it can generate examples.

Term Generation and Testing. Graffiti was a conjecture generation systems specifically for graph theory (Fajtlowicz, 1988). It maintained a library of example graphs, and attempted to generate conjectures of certain shapes, then checking them for consistency with its library examples. Unlike QuickSpec, where new examples can be generated on demand, Graffiti's set of examples was fixed, leading to production of relatively many non-theorems.

The systems IsaCoSy (Johansson et al., 2011) and IsaScheme (Montano-Rivas et al., 2012) are conceptually similar to QuickSpec, relying on term generation with different restrictions to avoid trivial and redundant conjectures, combined with random testing. A key difference is however that QuickSpec evaluates terms (and records the results), not whole equational statements, which greatly improves efficiency. Another closely related system from the functional programming community is Speculate (Braquehais & Runciman, 2017), which like QuickSpec, was designed to discover properties about Haskell programs.

Neural Networks. Recently, techniques using large neural network architectures from natural language processing have been applied to the problem of discovering new conjectures. Urban & Jakubův (2020) train the transformer model GPT-2 on the Mizar Mathematical Library. With the right parameters, they get GPT-2 to generate novel and well-typed conjectures in Mizar format.

However, the output may include duplicates from the library as well as non-theorems. Rabe et al. (2021) present a system based on self-supervised language models for mathematical reasoning tasks, including generating a missing precondition for a given conditional statement, generating one side of an equation given the other side, as well as "free-form" conjecturing. Between 13–30% of generated statements were both provable and new, with the remainder being for instance exact copies or alpha-renamings of statements from the training set, or simply false.

The neural network approach is of course radically different to QuickSpec. QuickSpec was designed to produce relatively small sets of interesting equations at a time, intended to be read by a human as a specification for a functional program. It rarely generates any false or duplicate conjectures, thanks to the integrated testing in the equation formation. Furthermore, QuickSpec is not dependent on training data about the theory at hand being available, but does on the other hand require generators to produce test data, with the caveat that the test data needs to be computable, or that a computable proxy can be used (as in the octonion example).

4 Further Work and Conclusions

The idea behind QuickSpec is simple: generate interesting terms and evaluate them on random test data to see which ones appear to be equal. Despite this, it is fast and efficient when given relatively small theories to explore. In a sense, QuickSpec is data-driven: it uses generator functions from the automated testing framework QuickCheck to generate and evaluate as many ground examples as it needs. This is also one of its limitations: trying to evaluate functions of high complexity (e.g. exponentiation in Peano arithmetic) can drastically slow down testing if a large test case is generated. Similarly, co-recursive functions can lead to non-terminating values. One solution is to extend QuickSpec with observational equivalence checking (Einarsdóttir et al., 2018). The other main weakness of QuickSpec is scalability. As it generates all non-redundant terms, the search space eventually becomes intractable when faced with large theories (say, a library with 30+ functions) or when asked to explore very large terms (sizes over about 9). Here, a combination with machine learning could be fruitful. A human user can immediately judge which of QuickSpec's equations are "nice and sensible" or, on occasion, "weird and ugly" and thus not very interesting. Certain shapes of conjectures are often both more useful and aesthetically pleasing, and QuickSpec could be steered towards that part of the search space first. A recent extension to QuickSpec experiments with doing this, using user provided templates when exploring larger theories and terms (Einarsdóttir et al., 2021). A natural next step is to instead attempt to learn templates from a library of lemmas, as suggested in Heras et al. (2013), thus creating a hybrid system where machine learning steers the term generation towards interesting parts of the search space. Learned templates may also facilitate search of non-equational conjectures, while still managing search space size.

We consider automated conjecture discovery a very interesting problem domain to tackle for researchers wishing to push the boundaries for AI in mathematics. While there are similarities between natural languages and formal languages, it is unlikely that AI methods for NLP will apply unmodified to mathematics. Specifically, huge amounts of natural language text data are available on the internet for big languages like English, while for mathematical languages, specific datasets must be collected and curated, as mathematical data typically appears in many different formats. It may be challenging to generalise well between systems, just as it is difficult to apply some NLP-ML methods successful on English to small spoken languages with little data. Furthermore, when starting a completely new mathematical formalisation (analogous to inventing a meaning for a bunch of new, previously unseen "words") no example data at all will be available. Here, we need the ability to explore and perhaps recognise analogies to other, known theories. We therefore speculate that a good solution to the problem of interesting conjecture invention will require a combination of both symbolic search-based methods, automated testing and machine learning.

References

John C. Baez. The octonions. *Bulletin of the American Mathematical Society*, 39:145–205, 2002.

R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, 1979.

Rudy Braquehais and Colin Runciman. Speculate: discovering conditional equations and inequalities about black-box functions by reasoning from test results. In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*, pp. 40–51, 2017.

Koen Claessen and John Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. In *Proceedings of ICFP*, pp. 268–279, 2000.

Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating inductive proofs using theory exploration. In *Proceedings of the Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pp. 392–406. Springer, 2013.

Simon Colton. The HR program for theorem generation. In Andrei Voronkov (ed.), *Automated Deduction—CADE-18*, pp. 285–289, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

Lucas Dixon and Moa Johansson. IsaPlanner 2: A proof planner for Isabelle, 2007.

Sólrún Halla Einarsdóttir, Moa Johansson, and Johannes Åman Pohjola. Into the infinite - theory exploration for coinduction. In *Proceedings of AISC 2018*, pp. 70–86, 01 2018. ISBN 978-3-319-99956-2. doi: 10.1007/978-3-319-99957-9_5.

Sólrún Halla Einarsdóttir, Nicholas Smallbone, and Moa Johansson. Template-based theory exploration: Discovering properties of functional programs by testing, 2021. Accepted for the post-proceedings of IFL'20, to appear.

Siemion Fajtlowicz. On conjectures of Graffiti. *Annals of Discrete Mathematics*, 38:113–118, 1988.

Jonathan Heras, Ekaterina Komendantskaya, Moa Johansson, and Ewen Maclean. Proof-pattern recognition and lemma discovery in ACL2. In *Proceedings of LPAR*, 2013. doi: 10.1007/978-3-642-45221-5_27.

Andrew Ireland and Alan Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16:79–111, 1996.

Moa Johansson. Automated theory exploration for interactive theorem proving: An introduction to the Hipster system. In *Proceedings of ITP*, volume 10499 of *LNCS*, pp. 1–11. Springer, 2017. ISBN 978-331966106-3.

Moa Johansson, Lucas Dixon, and Alan Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47(3):251–289, Oct 2011. ISSN 1573-0670. doi: 10.1007/s10817-010-9193-y. URL https://doi.org/10.1007/s10817-010-9193-y.

P. Jordan, J. v. Neumann, and E. Wigner. On an algebraic generalization of the quantum mechanical formalism. *Annals of Mathematics*, 35(1):29–64, 1934. ISSN 0003486X. URL http://www.jstor.org/stable/1968117.

Douglas B. Lenat. AM, an artificial intelligence approach to discovery in mathematics as heuristic search. 1976.

R. L. McCasland, A. Bundy, and P. F. Smith. MATHsAiD: Automated mathematical theory exploration. *Applied Intelligence*, Jun 2017. ISSN 1573-7497. doi: 10.1007/s10489-017-0954-8. URL https://doi.org/10.1007/s10489-017-0954-8.

Omar Montano-Rivas, Roy McCasland, Lucas Dixon, and Alan Bundy. Scheme-based theorem discovery and concept invention. *Expert systems with applications*, 39(2):1637–1646, 2012.

Markus N. Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. Mathematical reasoning via self-supervised skip-tree training, 2021. Accepted for ICML 2021, to appear.

Nicholas Smallbone, Moa Johansson, Koen Claessen, and Maximilian Algehed. Quick specifications for the busy programmer. *Journal of Functional Programming*, 27, 2017. doi: 10.1017/S0956796817000090.

Josef Urban and Jan Jakubův. First neural conjecturing datasets and experiments. In *Proceedings of CICM*, 2020.