Summary of Project

Krzysztof Drewniak, Henrik Bartels

8 Jan 2018

The aim of the project was to find a method to automatically generate the expression nomalization and/or property inference portions of Linnea from linear algebra knowledge in a known-correct fashion. We have investigated two possible approaches, neither of which yielded any immediately promising results: creating a term rewriting system using variants of Knuth-Bendix completion and integrating theorem provers into Linnea.

1 Rewrite systems

The first approach we investigated was generating a term rewriting system from the axioms of linear algebra and then completing it with a variant of the Knuth-Bendix algorithm. The concept behind this approach was to take the axioms of linear algebra and transform them into rewrite rules, such that, if two expressions rewrote to the same result under the rules, they were mathematically equivalent. The property that a set of rewrite rules must exhibit to fulfill this requirement is *confluence*: informally, that it doesn't matter which order the rules are applied in.

Overviews of the theory behind term rewriting systems, completion, and other such topics can be found in [3, 16].

Fortunately, the Knuth-Bendix algorithm, which is described in multiple sources such as [6] can take a system of equations and generate a confluent rewriting system from it. Unfortunately, the algorithm can fail or fail to terminate under some circumstances. The general concept of the algorithm is that you repeatedly find cases where confluence fails (to do this, you try to unify all the left-hand sides and then apply the rule(s) we derived the expression from) and then, after normalizing the results of the two rewrites, insert those normalized terms as a rewrite rule. This process in continued until (hopefully) there are no more rules to add, at which point the system is completed.

There are several subtleties that might be encountered while implementing this algorithm, such as needing to delete redundant rules, when new rules can be applied to existing rules, what order to search the possible extra rules in (the answer is "shortest expression first") and so on. Unfortunately, there don't seem to be any sources that present the algorithm in a clear, complete way, so this information needs to be cobbled together from various documents.

One important issue is the ordering needed for the algorithm. To turn an equation into a rewrite rule, we need a procedure for determining which way the arrow points. Some of these are described in the books referenced above, as well as in, say [5]. A relevant one for this project was the lexicographical path ordering, which, roughly, takes a total ordering that tells you which operators/constants should be "further in" the expression. Recent projects like Slothorp[19] (along with older projects like [7]) have tried to autogenerate the ordering, but those didn't work in relevant cases (like rings). There was also a method to nondeterministically find a working ordering [15], but I didn't look into it much.

The Knuth-Bendix algorithm is, in its unmodified form, not suitable for linear algebra's axioms. One major issues arises with commutativity, which is an axiom than generally can't be oriented. Fortunately, there are solutions for this. The general concept is to perform the completion modulo commtativity (or, more typically associativity and commutativity, giving "AC" completion). Algorithms for this can be pieced together from a few sources [4] gives a good overview of AC rewriting and other extensions, as do the usual books, while [14] gives an effectively complete (although hard to understand) algorithm. The Peterson paper [14] also has several examples. It should be noted that, based on this paper, the code we have implement for Knuth-Bendix completion modulo AC (of +, in our case) has a bug that we have not been able to correct (good luck).

There are two main tricky parts to AC rewriting as it's typically implemented. The first is that, for many rules, you need extended versions, where you add dummy arguments to the top level of the rule so that instances of the rule don't get skipped because pattern-matching doesn't work (managing this is nonobvious). The second is associative-commutative unification. This is, it should be noted, an NP-complete problem. The always-correct but slow algorithm is [18] (the underlying basis-finding algorithm is [10]), while the never-*incorrect* algorithm we implemented is [12]

The next step after associative-commutative rewriting was figuring out how te integrate conditional rules, such as $A^T = A$ if A is symmetric. We took a look at many-sorted or order-sorted completion [8, 9], but it didn't have the power needed to express things like AB is lower triangular if A and B are (it'd've been more appropriate for the case where there are several distinct types of things, like scalars, vectors, and matrices) Various conditional rewriting schemes found in the books and [13] didn't have the power for the sort of back-and-forth we were wanting either. The straightforward step of doing things like $A^T \to \text{issymmetric}(A, A)$ quickly resulted in systems that were impossible to consistently order, and so was useless in practice. This showed that the conditional parts of the linear algebra axioms were a major barrier to making the rewrite-rule approach work.

Another issue was the fact that matrix multiplication is associative but not commutative. Associativity sort of works as a rewrite rule such as $(xy)z \to x(yz)$, but this is known not to capture all equivalences in some cases. Treating associative operators like AC ones (rewriting modulo them) runs into the problem that there can be infinitely many ways to unify two expressions involving an associative-only operator where one isn't a specialization of another. The main example is that, with associative $\cdot$ and constant a, xa and ax can have the substitutions $\{x \to a\}, \{x \to aa\}, \{x \to aaa\}$ and so on, where none of those can be transformed into any of the other ones with a substitution.

These two issues combined to make the Knuth-Bendix-based approach infeasible, and so it was abandoned.

2 Theorem provers

Since rewrite rules weren't working, we decided to turn to various automated and semi-automated theorem provers. The idea was to call out to these programs to prove some combination of the properties of expressions (under given input assumptions about matrices) or the equivalence of expressions. As an initial test-case, we tried to implement some very basic property inference.

One program we investigated was E (`http://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html`), which implements first-order logic with equality.

The input format is TPTP (`http://www.cs.miami.edu/~tptp/`), which is rather poorly documented. The drawbacks of E were that it does not always terminate on false theorems, there seems to be some issue that results in non-theorems being true under the axioms we developed, and that the system does not implement the natural numbers natievly, which means they would need to be constructed if needed. E is, however, rather fast, and it has a server mode that would have been quite useful in Linnea.

The other tool we investigated was Lean [2, 1]. Lean is a dependently-typed "programming language" that proves theorems but is rather awkward to use in practice at this time. This meant that efforts to formalize property inference in Lean were rather unsuccessful. (Lean does also have a server mode, as seen is [11]). It should be noted that the Centigrad project[17] has successfully used Lean to write formally-correct machine-learning algorithms in the style of Tensorflow, and that their approach and/or code might be useful for our use-case eventually.

Much of the software we investigated was not yet suitable for our purposes or inconvenient to use, so the theorem-prover approach has also not yielded useful results.

References

[1] Jeremy Avigad, Leonardo De Moura, and Soonho Kong. "Theorem Proving in Lean". In: (2017).

[2] Jeremy Avigad et al. "An introduction to Lean". 2017. URL: `https://leanprover.github.io/introduction%7B%5C_%7Dto%7B%5C_%7Dlean/introduction%7B%5C_%7Dto%7B%5C_%7Dlean.pdf`.

[3] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Vol. 145. 1. 1998, p. 40. ISBN: 978-0521779203. URL: `http://books.google.com/books?hl=de%7B%5C&%7Dlr=%7B%5C&%7Did=N7BvXVUCQk8C%7B%5C&%7Dpgis=1`.

[4] Leo Bachmair. *Canonical Equational Proofs.* Birkhäuser Basel, 1991. ISBN: 9780817635558.

[5] Nachum Dershowitz. "Termination of rewriting". In: *Journal of Symbolic Computation* 3.1-2 (1987), pp. 69–115. ISSN: 07477171. DOI: `10.1016/S0747-7171(87)80022-6`.

[6] A J J Dick. "An Introduction to Knuth-Bendix Completion". In: *The Computer Journal* 34.1 (1991), pp. 2–15. ISSN: 0010-4620. DOI: 10.1093/comjnl/34.1.2. URL: http://comjnl.oxfordjournals.org/content/34/1/2.abstract%7B%5C%%7D5Cnhttp://comjnl.oxfordjournals.org/content/34/1/2.full.pdf%7B%5C%%7D5Cnhttp://comjnl.oxfordjournals.org/cgi/content/abstract/34/1/2.

[7] Jeremy Dick, John Kalmus, and Ursula Martin. "Automating the Knuth Bendix ordering". In: *Acta Informatica* 28.2 (1990), pp. 95–119. ISSN: 00015903. DOI: 10.1007/BF01237233.

[8] Harald Ganzinger. "Order-sorted completion: the many-sorted way". In: *Theoretical Computer Science* 89.1 (1991), pp. 3–32. ISSN: 03043975. DOI: 10.1016/0304-3975(90)90105-Q.

[9] J . A Goguen and J Meseguer. "Completeness of Many-Sorted Equational Logic". In: *Houston Journal of Mathematics* 11.3 (1985), pp. 307–334.

[10] Dallas Lankford. "Non-negative Integer Basis Algorithms for Linear Equations with Integer Coefficients". In: *Journal of Automated Reasomng* 5 (1989), pp. 25–35.

[11] Robert Y Lewis. "An extensible ad hoc interface between Lean and Mathematica". In: (2017). ISSN: 2075-2180. DOI: 10.4204/EPTCS.262.4. arXiv: 1712.09288.

[12] Patrick Lincoln and Jim Christian. "Adventures in associative-commutative unification". In: *Journal of Symbolic Computation* 8.1-2 (1989), pp. 217–240. ISSN: 07477171. DOI: 10.1016/S0747-7171(89)80026-4.

[13] Massimo Marchiori. "On deterministic conditional rewriting". In: *Mit Laboratory for Computer Science* 405 (1997). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.5251.

[14] Gerald E. Peterson and Mark E. Stickel. "Complete Sets of Reductions for Some Equational Theories". In: *Journal of the Association Computing Machinery* 28.2 (1981), pp. 233–264. ISSN: 0004-5411. DOI: 10.1145/322248.322251.

[15] David A Plaisted. "A simple non-termination test for the Knuth-Bendix method". In: *International Conference on Automated Deduction.* 1986.

[16] David A. Plaisted. "Equational reasoning and term rewriting systems". In: *Handbook of logic in artificial intelligence and logic programming (vol. 1)* (1998), pp. 274–364.

[17] Daniel Selsam, Percy Liang, and David L. Dill. "Developing Bug-Free Machine Learning Systems With Formal Mathematics". In: (2017). arXiv: `1706.08605`. URL: `http://arxiv.org/abs/1706.08605`.

[18] Mark E. Stickel. *A Complete Unification Algorithm For Associative-Commutative Functions*. Tech. rep.

[19] Ian Wehrman, Aaron Stump, and Edwin Westbrook. "Slothrop: Knuth-Bendix completion with a modern termination checker". In: *Term Rewriting and Applications* 4098 (2006), pp. 287–296. ISSN: 16113349. DOI: `10.1007/11805618_22`. URL: `http://link.springer.com/chapter/10.1007/11805618%7B%5C_%7D22`.