Proof Checking for Mathematical English

Christopher A. Stone

January 10, 2014

Abstract

I am requesting funding for a summer research student. The topic is a software system to automatically verify the correctness of mathematical proofs, and more specifically, to verify proofs written in the stylized English of mathematical textbooks, papers, and monographs. Essentially, the goal is to go beyond spell-checking and grammar-checking to logic-checking.

Since this would be the start of a new project, the immediate goal for the summer would be an exploratory prototype, flexibly handling natural-language proofs for the limited domain of propositional (boolean) logic. My hope is that this will kick off a longer-term research project; at the very least, the results will make a good CS 81 demonstration.

Motivation

Proofs are a key foundation for mathematics and computer science, but even careful reasoners make mistakes, and even careful reviewers can miss them. Automation seems a natural solution. With general-purpose theorem-proving still an unsolved problem, a more practical approach is used by *proof assistants* such as Coq or Twelf. These software tools help humans construct formal proofs that can be automatically checked (i.e., that each step follows from the preceding ones), and they are seeing increasing use. In programming language research, many authors provide machine-checked proofs of their theory results (including all the bits that are too repetitive or to boring to fit into their papers), and there has been impressive success in using proof assistants to prove the correctness of very large programs.

Unfortunately, there is a disconnect between the proofs that are published and the proofs that are being checked. As just one example, the recently-published book *Homotopy Type Theory* is available in two formats: a traditional mathematics text (in LaTeX) with explanations, definitions, theorems, and proofs; and proof scripts for the Coq proof assistant. Since the proof scripts have been mechanically checked for correctness, one might expect the book to have no technical errors. But the number of bug reports (errata) continues to slowly increase, and not all of these are simple typos. The problem is that the book proofs and the Coq proofs were written independently, often by different people.

Why not just put the Coq proofs into the book? When using a proof assistant it is often much easier to use methods that would be considered "unnatural" or "ugly" when writing a proof intended for other people; if the computer will automatically do complex algebraic simplifications or exhaustively check hundreds of cases in microseconds, there's little need to be clever or elegant.

More importantly, Coq has been accurately described as "write-only." A Coq proof script is a sequence of *tactics*, each of which describes not the proof itself, but rather an action taken by the user while constructing the proof; the result is a list of verbs without context. For example, here is a proof script for the fact that for any two integers n and m, if $-n = -m$ then $n = m$:

```
intros n m;  destruct n, m;  simpl;  intros H;  destr_eq H;  now f_equal.
```

It doesn't help to know that, e.g., `simpl` means "simplify the proof goal as much as possible," or that `f_equal` means "use the fact that applying a function to equal arguments gives equal results." What simplifications actually happened? What function was being applied to equal arguments? The only way to find out is to replay the script within the interactive system, step-by-step. There is no insight to be gleaned from the static page.

Other systems like Mizar attempt to make proof scripts more readable, as in the following excerpt from a proof that there are infinitely many primes:

```
 consider p such that
A1: p is prime & p divides k by INT_2:48;
A2: p <> 0 & p > 1 by A1,INT_2:def 5;
 take p;
 thus p is prime by A1;
 assume p <= n;
 then p divides n! by A2,NAT_LAT:16;
 then p divides 1 by A1,NAT_1:57;
 hence contradiction by A2,NAT_1:54;
```

but this is still not nearly as intelligible as a typical book proof, and is tedious to write. What we would prefer is a system that can directly understand and verify proofs like:

Theorem 1 (Peirce's Law). $((p \to q) \to p) \to p$

Proof. Assume that $(p \to q) \to p$. It suffices to show p, and we can do so by contradiction. Suppose $\neg p$. Then p is false and thus implies q. But then our original assumption says that p follows, and we have our contradiction. $\square$

The Project

I would like funding for a summer research student to help me investigate the problem of checking natural-language proofs directly.

The summer would start with looking into existing NLP tools already available (such as NLTK) and existing lexicons and grammars for English, to avoid starting completely from scratch. We would also be collecting sample proofs, to study how different authors express the underlying logical structure. Then we would start with a simple prototype system that translates a small subset of English into a machine-checkable representation (e.g., the natural deduction symbolic proof style used in CS 81, or inputs that can be handled by a proof assistant such as Coq). Finally, we would iteratively improve and extend the prototype: handling a larger range of inputs, generating better outputs, improving the user interface, and so on. The student would be deeply involved in all aspects of background research, system design, coding, and testing.

There are several reasons this project seems feasible. First, a significant inspiration for this project is Mohan Ganesalingam's 2013 book *The Language of Mathematics*[1]. It makes a compelling argument that language used in mathematical proofs is unusually well-suited to natural language processing (NLP) algorithms and techniques. Their syntax is comparatively simple, and the well-defined semantics of mathematics provides context when sentences are semantically ambiguous.

Second, it should be easy to find an interested student with the right background. We have a lot of students interested in Artificial Intelligence (our AI elective is invariably oversubscribed), and logic and proofs are familiar from math and CS required courses.

Third, while NLP is not my usual research area, I do have experience with logic, proofs, and the (somewhat simpler) syntax and semantics of computer languages. There are a number of open-source NLP resources available, and I also have acquaintances who work in the NLP domain. In particular, Kim Bruce at Pomona (who has taught both CS 81 and a computational linguistics elective) has agreed to serve as a consultant for this project.

Finally, even a simple prototype that can handle a small subset of English for propositional logic proofs would make a useful demonstration in CS 81. When we study natural deduction, I argue that the formal rules are a codification of the logical steps taken in traditional mathematical proofs; having a system that can demonstrate this correspondence would be even better. (And I believe we could accomplish more than just this minimal prototype.)

The project is potentially very open-ended, and if the summer goes well I would anticipate looking for external funding to continue the project.

[1]The other main inspiration is the frustrating experience of trying to make sense of the Coq proofs for the Homotopy Type Theory book!