Joshua Tauber

Week 5 response

My area of research is in formal methods for verification of distributed computation. One of the chief goals of our research group (and my research program, in particular) is to increase the automation of reasoning about the correctness of distributed system designs and their implementations.

One of the chief tools we use is the Larch Prover (LP). LP is an automated proving assistant. In the Donald MacKenzie's taxonomy, LP is an interactive prover that produces non-human-like proofs. Producing a "proof" with LP generally consists of an iterative process very much like the one described Boyer. The human enters a conjecture which LP then attempts to prove. LP will apply one or more of its available rules to generate (hopefully simpler) subgoals to prove. Most likely LP will get stuck somewhere in the process and need guidance to continue. For example, the user might have to suggest a value to instantiate into an existential quantifier in order to discharge a branch of a case analysis. The result at the end of a successful interaction is a claim by LP that the desired conjecture has been proved.

The interesting point is to examine what remains from this process. The LP user now has a "script" for proving the theorem. The script is the series of commands that the user enters to get LP to agree that the conjecture is true. Entries in the script may include some obvious steps that any mathematician will recognize (at least once they are decoded) like "use induction on A" or "instantiate B as D+E". On the other hand, the script may include commands asking LP to perform larger operations like "use critical pairs" (an operation related to the resolution). Most interesting however, is what is _not_ there. What is not there is the actual text of what would recognized as a formal proof. That is, the proof script consist _only_ of the hints (commands) to give LP when it gets stuck.

To be fair, one could ask LP to print out most of the intermediate steps in excruciating detail. In practice, however, no one ever does. Why? For three reasons.

First, the details of the formal proofs are not actually very interesting. The exact steps do not serve an explanatory function. This is exactly the same reason almost no full text formal proofs have been published. Mathematicians care that the formal steps _can_ be done, but they do not care what they actually are.

Second, partly by design, LP tends not to jump over the "big steps" on its own. So the interesting bits of the proof (e.g. its structure) do tend to end up in the proof script. If you are lucky, the interesting bits are not swamped by extraneous minutia.

Third, we trust LP to do the brute force plodding flawlessly. LP may not be very "smart". It probably won't get to the end of a proof on its own. However, the steps it does take we trust to be sound.

In many ways, this third point distinguishes the whole research area of theorem provers and automated theorem proving assistants from the sort of computer assisted proof done by Appel and Haken. In the proof of the four color theorem, Appel and Haken used a specialized (and if my old adviser, David Gries, is to be believed, poorly structured) computer program to verify certain cases of the proof. The very fact that this program was written as part of the proof makes verifying the program part of verifying the proof.

In contrast, theorem provers are used, examined, extended, and debugged repeatedly over a period of years. The very fact that a community of users exists for a particular tool lends assurance that the tool performs its assigned tasks as requested. In some sense, the correctness of the theorem prover itself is a conjecture that is separable from the theorems it is used to check. Does this body of evidence for the correctness of a theorem prover actually constitute a proof, in the mathematical sense? Perhaps not. But by the same token, neither does peer review and publication of a purported "proof" constitute definitive proof that the publication is, in fact, correct.

In some ways LP proof scripts are easier to check than large mathematical treatise. I suspect, for example, that it would be easier me to write my own theorem proving assistant to recheck an LP script than for me to verify the steps in Andrew Wiles proof of the Taniyama-Shimura conjecture.

Do LP proofs scripts serve explanatory functions? They may. Over time our research group has developed a body of proof technology in a mathematical sense. As a result, the proofs about distributed system that our group produces follows a very stylized form. (This stylized form is what has led to the push for automation of the proof techniques.) As a result, the part of the proof then ends up in the LP proof script tends to be the part that makes this particular proof different from previous ones. Thus, to the educated eye, the proof script actually highlights the interesting parts of the proof.

I would argue, however, that not all proofs need be explanatory. Some are in fact, practical ends in themselves. That is, proving a system correct may only be as interesting in so far as one cares that the system is correct and it may have no further implications for building further mathematical structures.
In these cases, proof scripts become very useful. In particular, when a system design changes it is often very easy to reverify the system using LP when reworking even an informal proof by hand may be extremely cumbersome.