Math.Comput.Sci. (2015) 9:1–3
DOI 10.1007/s11786-014-0214-9

Mathematics in Computer Science

Foreword to the Special Focus on Formal Proofs for Mathematics and Computer Science

Laurent Théry · Freek Wiedijk

Published online: 28 October 2014

Formal Proof: State of the Art

The art of *formalization* consists of writing a proof in an artificial mathematical language in such a way that a computer program can construct and validate a formal version of that proof that is completely certain to be free of errors. These programs are called proof assistants or interactive theorem provers. Formalization has both applications in mathematics and in computer science.

Note that this quest for certainty comes at a price. First, there are some foundational issues. Does the logic of the system "make sense", i.e. is it consistent? Is it powerful enough, i.e. can we derive a proof of any true statement? These problems were at the root of the creation of a new field of mathematics, Mathematical Logic, at the beginning of the 20th century, with key contributions by Frege, Russell and Gödel among others. Second, the system on which the proofs are done is a computer system and any bug in this system taints the benefit of having a formal proof. Finally, one needs to take special care of the formal definitions that have been introduced. They have to match the informal understanding of the notions that the proof is about. Consider, for example, the simple problem of formally proving that there can be at most three Friday the 13ths in a single year. Central to this proof will be to correctly formalize what a year is.

In recent years highly non-trivial proofs have been formalized, showing the power of current formalization technology. Examples are, in mathematics:

- Harrison's formal proof of the Prime Number Theorem [7];
- Gonthier's formal proofs of the Four Color Theorem [4] and the Feit-Thompson Theorem [5];
- Hales' formalization of his proof of the Kepler conjecture [6] in the Flyspeck project [15].

L. Théry (✉)
Inria Sophia Antipolis-Méditerranée, 2004, route des Lucioles-BP 93,
06902, Sophia Antipolis Cedex, France
e-mail: Laurent.Thery@inria.fr

F. Wiedijk
Institute for Computing and Information Sciences, Radboud University Nijmegen,
Toernooiveld 212, 6525 EC, Nijmegen, The Netherlands
e-mail: freek@cs.ru.nl

and in computer science:

- Fox's formal proof of the correctness of the ARM6 architecture [2], followed by his work [3] on the L3 specification language that has been used to formally specify many instruction set architectures;
- Leroy's formal proof of the correctness of an optimizing C compiler [10] in the CompCert project [1];
- Klein's formal proof of the correctness of the seL4 microkernel operating system, as developed in the L4.verified project [8];
- Myreen's formal proof of the correctness of the CakeML compiler [9], on top of which Kumar is constructing a formal proof of the correctness of a HOL based interactive theorem prover [14] similar to the system in which the formal proofs have been created.

As can be seen from these examples, interactive theorem provers are now very powerful. Among the different flavors of interactive theorem proving, the most popular systems are Coq, Isabelle, HOL4 and HOL Light. Coq is based on *type theory*, while the other three are closely related and are based on *higher order logic*.[1]

These projects all show that the technology of formal proof has recently matured significantly. Turning a mathematical or computer science proof into a formal proof now is routine for experts. Given an informal proof that is sufficiently correct, formalizing it is a matter of putting in enough man-hours.

The research community working on formal proof is currently relatively small, and interactive theorem provers do not have many users yet. It is an interesting question what causes this lack of attention, given the power of the technology. One of the reasons, maybe the foremost reason, is that knowing that a proof is completely correct is often not considered the most important thing. For instance in computer science, most systems have a large number of known "bugs", and it is not considered economically interesting to spend the money on removing more than the most essential of those. Similarly, in mathematics, when faced with the choice between pursuing new mathematics and making sure that all the details of a proof are correct (where "mistakes" that are discovered in a proof during an attempt to get everything correct often are easily fixable), many mathematicians consider pursuing new mathematics to be a more valuable use of their time. Of course this situation may change in the future. While a decade ago computing was confined to dedicated areas, it is now diffusing to all aspects of human activity. Our society is depending more and more on software-based systems and getting highly-reliable systems clearly is a must. Also, for mathematics, theorem proving systems can be used as a tool to explore new territories. The Flyspeck project shows how they can be used to safely incorporate computations into proofs while the Feit-Thompson project shows how they can be used to build collaborative proofs and safely glue together proofs that require strong expertise in different areas of mathematics.

Even if one considers it worthwhile to get a proof exactly right, there are three main reasons why the cost/benefit ratio of formalizing that proof currently is so high that only a few people take the effort to use this approach.

- Interactive theorem proving has a steep learning curve. Not only has one to be trained in logic, but also there is a fundamental difference between an informal proof and a formal proof. In some sense, the situation is similar to programming where knowing an algorithm is often not sufficient to implement it on machine. So there is a clear need for more students to be trained at university level. Unfortunately with some noticeable exceptions such as Concrete Semantics [11] and Software Foundations [13], not enough training material is currently available.
- Second, when creating a formal proof, one currently needs to be aware of many details of the proof. Although the best systems have implementations of various decision procedures and proof search procedures, the level of detail of a formal proof generally still is higher than in traditional presentations. Luckily, the automation of the systems is steadily improving. In particular, various researchers are pursuing "hammers", proof procedures similar to Isabelle's Sledgehammer [12] that apply machine learning techniques to be able to automate proof steps that go beyond the specific domains of decision procedures and proof search restricted to first order predicate logic.

[1] The ACL2 system has also been used for very impressive projects. It is based on a version of *primitive recursive arithmetic*, which is less well suited for proofs in abstract mathematics.

– Another difficulty is that currently there are no good comprehensive libraries of formal mathematics to build on. Many systems collect their formal proofs in archives (like Coq's contribs, or Isabelle's Archive of Formal Proofs), but these libraries are then generally not organized, cleaned and improved, to get a proper basis for further formalization. A few libraries attempt to go further than this. Most notably there is Gonthier's Mathematical Components library, but there are also the Mizar Mathematical Library, and HOL Light's library of mathematics. These libraries are better organized, but still suffer of two main problems: one structural, the other fundamental. First, the best libraries generally have been created from the vision of one person, who kept tight control over the development of that library. If one compares this with efforts like the free software community or the development of Wikipedia, it seems that the current development of formal libraries does not scale as well. Second, these libraries often suffer from idiosyncrasies of their system. For example, in Coq one needs to work around a lack of extensionality and proof irrelevance, and in the HOL based systems a formal treatment of some subject often is lacking due to the fact that the system does not have dependent types. While proposals like Vovoedsky's *homotopy type theory* (HoTT) [16] may improve the situation in the case of Coq, the system where formalizing mathematics can be done seamlessly is yet to be built.

Content of the Focus

Three interesting contributions compose this focus. They address two of the three bottlenecks that were listed above. HOL(y)Hammer proposes a "hammer" for the HOL Light system which can prove many lemmas from the Flyspeck library without any human help. The Graph Library and Coquelicot are a start of comprehensive libraries of certain basic mathematical notions.

References

1. CompCert. http://compcert.inria.fr/
2. Fox, A.C.J.: Formal specification and verification of ARM6. In: TPHOLs'03, LNCS, pp. 25–40 (2003)
3. Fox, A.C.J.: Directions in ISA specification. In: ITP'12, volume 7406 of LNCS, pp. 338–344 (2012)
4. Gonthier, G.: Formal proof—the Four Color Theorem. Not. AMS **55**(11), 1382–1393 (2008)
5. Gonthier, G., et al.: A Machine-checked proof of the odd order theorem. In: ITP'13, volume 7998 of LNCS, pp. 163–179 (2013)
6. Hales, T., Harrison, J., McLaughlin, S., Nipkow, T., Obua, S., Zumkeller, R.: A Revision of the Proof of the Kepler Conjecture. In: The Kepler Conjecture, pp. 341–376. Springer, New York (2011)
7. Harrison, J.: Formalizing an analytic proof of the prime number theorem (dedicated to Mike Gordon on the occasion of his 60th birthday). J. Autom. Reason. **43**, 243–261 (2009)
8. Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: formal verification of an operating-system kernel. Commun. ACM **53**(6) (2010)
9. Kumar, R., Myreen, M.O., Norrish, M., Owens, S.: CakeML: a verified implementation of ML. In: POPL, pp. 179–192. ACM Press, New York (2014)
10. Leroy, X.: Formal verification of a realistic compiler. Commun. ACM **52**(7) (2009)
11. Nipkow, T., Klein, G.: Concrete semantics with Isabelle/HOL. http://www21.in.tum.de/~nipkow/Concrete-Semantics/
12. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011, vol. 2 of EPiC Series, pp. 1–11. EasyChair (2010)
13. Pierce, B.C., Casinghino, C., Gaboardi, M., Greenberg, M., Hriţcu, C., Sjöberg, V., Yorgey, B.: Software Foundations. http://www.cis.upenn.edu/~bcpierce/sf/current/index.html
14. Ramana Kumar, M.O.M., Arthan, R., Owens, S.: HOL with definitions: semantics, soundness, and a verified implementation. In: ITP'14, vol. 8558, pp. 308–324. LNCS (2014)
15. The Flyspeck Project. https://code.google.com/p/flyspeck/
16. The Univalent Foundations Program. Homotopy type theory: univalent foundations of mathematics. Institute for Advanced Study. http://homotopytypetheory.org/book (2013)