*Under consideration for publication in Math. Struct. in Comp. Science*

# Zen and the art of formalization

ANDREA ASPERTI[1] and JEREMY AVIGAD[2]

[1] *Department of Computer Science, University of Bologna, Italy.*
[2] *Departments of Philosophy and Mathematical Sciences, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA.*

*Received 23 November 2010*

> *"If you can't explain your mathematics to a machine it is an illusion to think you can explain it to a student."*
> – N. G. de Bruijn[†]

N. G. de Bruijn, now professor emeritus of the Eindhoven University of Technology, was a pioneer in the field of interactive theorem proving. From 1967 to the end of the 1970's, his work on the Automath system introduced the architecture that is common to most of today's proof assistants, and much of the basic technology. But de Bruijn was a mathematician first and foremost, as evidenced by the many mathematical notions and results that bear his name, among them de Bruijn sequences, de Bruin graphs, the de Bruijn-Newman constant, and the de Bruijn-Erdös theorem. The quotation above is thus interesting not because it reflects on his expertise in formal verification, but, rather, his convictions as a working mathematician.

But what could he have meant? Although he spoke of "explaining" mathematics to a machine, it does not seem likely that he meant to presuppose that a machine is capable of *understanding* in any robust sense; and even less likely that he would have held that the ability to check the details of a formal proof is constitutive of mathematical understanding. To be sure, the ability to read a proof and ascertain its correctness is an important part of understanding it, but it is certainly not the only part. And it is by no means obvious that checking ordinary mathematical proofs has anything to do with matching patterns and checking rules in a formal symbolic calculus. Nor would de Bruijn have failed to notice that "explaining" mathematics to a machine is very different from explaining it to a student. It is no more obvious that formalizing mathematics to be checked by a computer has anything to do with explaining a piece of mathematics to a fellow human being.

So what was de Bruijn getting at? The quotation invites us to reflect on the nature of

[†] from "Memories of the Automath Project," an invited lecture at the *Mathematical Knowledge Management Symposium*, Heriot-Watt University, Edinburgh, Scotland, 2003.

formal mathematics, and its relationship to informal mathematics. In doing so, we come to realize that the very practice of formalization requires a deeper reflection on the logical and linguistic mechanisms that govern the representation of informal mathematical knowledge, and the way that such knowledge is organized as a structured collection of interconnected notions. Indeed, formalization is valuable because it *forces* us to think about mathematical knowledge in this way.

Mathematics is a precise discipline, but mathematical rigor is not the same as logical formality. When one tries to translate a mathematical argument to a format suited for automated processing, one realizes that there is a tangible mismatch between a textbook presentation and what is actually required by the machine. Ordinary mathematics is simply too informal for the logical and algebraic procedures that contemporary automated reasoning has to offer. At a linguistic level, ordinary mathematical discourse systematically overloads symbols and abuses notations in ways that make mechanical interpretation difficult. Subtle contextual cues are needed to resolve the ambiguities that are intrinsic to mathematical vernacular, requiring not just knowledge of the notation and conventions at play but, moreover, an understanding of the relevant mathematical discipline.

The situation is similar at the level of proofs. The difficulty in formal verification does not lie solely in the sheer number of formal logical steps needed to justify a single textbook inference. The problem is, rather, that ordinary mathematical proofs rely on patterns of reasoning—say, combinatorial, geometric, or diagrammatic—that are often difficult to analyze and explain in terms of formal logical inferences. Once again, the need to do so—that is, the process of figuring out how familiar patterns of mathematical inference can be expressed in formal terms—can provide valuable insights into the nature of the mathematics, and the nature of mathematical thought itself.

In short, the mismatch between the human activity known as "mathematics" and mechanical representations thereof is interesting in and of itself. Formalizing a piece of mathematical reasoning is like looking at it through a magnifying glass, trying to discover the complex patterns and mechanisms that lie beneath the surface. It is not just a matter of foundational reduction, that is, trying to reduce mathematical reasoning to a small number of basic principles and rules. Just the opposite: it is the holistic—one might say, phenomenological—goal of understanding how different components of mathematics are represented, how they fit together, how we interact with these representations, and how they shape our mathematical experience. Twentieth century foundational research suggests that mathematical inference can be reduced to formal inference, in principle; but understanding how this can be done *in practice* requires a reflective attentiveness to our everyday mathematical experiences. Such a philosophical rejection of the logistic bias is reflected in de Bruijn's comments on the architectural design of Automath:

> "Don't put logic into the system; let the user start his book with it. Don't put induction and recursion in the system; consider it as book material, even when that might be slightly clumsier."

The differences between foundational reduction and formal verification become evident, for example, in the treatment of definitions and lemmas. From the point of view of

axiomatic deduction, what is important about definitions and lemmas is simply that they can be eliminated: definitions can be expanded, and auxiliary lemmas can be replaced by their proofs, in such a way that inferential validity is preserved. This obscures the reason that definitions and lemmas are useful in the first place, namely, in structuring our knowledge and helping us to think and reason more efficiently. In contrast, formal verification focuses on precisely these features. In carrying out a formalization, we come to better understand the reasons that mathematical notions emerge and force themselves upon our attention. Ultimately, what we seek is an understanding of mathematical proof that can pave the way to a better *qualitative* investigation of mathematics, one that can help us understand what makes concepts fruitful, theories powerful, and proofs elegant or insightful. We are currently far from this goal, but obtaining better formal representations of mathematical knowledge is an important first step.

With the advent of interactive proof assistants, we are now beginning to develop substantial corpora of machine-readable mathematics. We take the fact that these corpora provide enough information for a machine to verify their correctness to be an indication that something of the relevant content of the mathematics has been captured. But one may wonder whether we have actually captured too much, in embedding our informal mathematical vernacular in the idiosyncrasies that are typical of foundational dialects. The substantial lack of interoperability between proof assistants, and the difficulties we face in translating information between the different foundational representations, seem to corroborate this concern.

The translation from high-level mathematics to a low-level foundational language is often compared to compilation from a high-level programming language to assembly language. Whereas high-level programming languages provide platform-independent ways of describing an algorithm, an assembly language is tied to a particular choice of processor. But extensive research on compilers has resulted in consolidated and highly modular architectural design, with clear interfaces and well-defined intermediate languages. Indeed, the introduction of platform-independent intermediate representations, which serve to improve portability and to reduce the cost of maintenance, represents one of the field's major breakthroughs. When it comes to the formalization of mathematics, we do not have anything comparable. This is all the more surprising given that so much of the time invested in formal verification comes in a preliminary phase in which one analyzes the given piece of mathematics and begins to transform it into something more suitable for formal encoding. That is, the result of this preliminary phase is a semi-formal version, in which one has rendered the concepts more precise, carefully mapped out the structure of the development, and begun to outline the details. All this typically takes place before one has written a single line of low-level "code." The fact that such a presentation is largely independent of the underlying formal axiomatic system suggests that it might be possible to develop more expressive languages with this same character. Indeed, the task of designing intermediate languages that are capable of expressing formal features of the mathematics while remaining independent of an axiomatic framework seems to be one of the main challenges to formal verification today.

The papers in this special issue can be viewed, narrowly, as describing various contributions to the corpus of formally verified mathematics, the technology that supports such

verification, and the theory that supports the technology. But let us not lose sight of the big picture: by clarifying what is needed to render ordinary mathematics in formal terms, such work constitutes an important contribution to our understanding of mathematics itself, much in the spirit of de Bruijn's words.