Lecture 27 – Finding Computational Content and Course Summary

Abstract

This lecture will illustrate finding computational content from classical proofs. Then it will summarize some of the main themes of the course and discuss the nature and future of proof assistants.

This lecture will help students relate their projects to the main themes of the course. Doing that will be a key factor in grading the projects as we discussed.

1 How we know mathematical truths

We have seen that there are at least two ways we come to know mathematical truths. One way is based on *computational experience*. We learn to grasp the idea of number by learning to compute with representations of numbers and relate these representations. We learn to grasp the idea of a function by knowing how to computationally transform input data to output data. We understand inductive reasoning by grasping the recursive functions that create evidence step by step, a "time" dependent process. This process becomes more subtle when we try to grasp the "fullness of time" in the sense conveyed by the concept of *continuous processes*, leading to the notion of the *continuum of real numbers*.

We also understand another way of knowing that is less concrete and depends on generalizing the idea of building evidence to building what we have called in Lecture 26 *virtual evidence*. That notion helps us understand the method of "classical" reasoning that emerged in the 1800's. That dominant style led to important distinctions in the way we organize mathematical knowledge and explain its relationship to computation. The need for this understanding has grown in importance as we build hardware to help us compute and programming languages to convey the algorithms, procedures, and heuristics that machines execute.

Proof assistants were built to help people solve mathematical problems, write correct programs, and grasp in fine detail the computational processes and virtual computations that underlie modern mathematics. Thereby a science of computation emerged grounded in machines that help us reason as well as compute. We call the field computer science, and one of its enduring goals has been to extend the reach of computation in nearly all aspects of life, especially to understanding the nature of mind and intelligence – both human and machine intelligence.

Some mathematicians and computer scientists believe that proof assistants are transforming not only how people work but also the very notion of knowledge itself. It might not be long before we must cope with a body of knowledge that is only accessible to humans via proof assistants. We can easily imagine elements of our knowledge that humans cannot manage alone. How large a step is it to imagine states of knowledge useful to machines but essentially inaccessible to unaided human understanding yet essential to technological progress and even to our survival? We will briefly consider these issues.

1.1 Finding and using the computational content of mathematical knowledge

Next we will consider examples of mathematical results that are manifestly computational and those for which the computational content must be "extracted" as well as those for which the computational content is only *virtual.* Some of the most creative and interesting explorations with proof assistants have been finding the computational content in cases where it is hidden or hard to find or not required because virtual evidence is sufficient in an initial effort to explore a topic. Logicians and computer scientists have developed ingenious methods to ferret out computational content using insights from the structure that emerges when the ideas and theorems are made completely formal.

We will examine a small example studied by the author and Dr. Chetan Murthy in our article *Finding Computational Content from Classical Proofs* [10]. The article is included on the course web page with this lecture.[1]

Consider the following claim:

$$\forall f : \mathbb{N} \to \{0,1\}.\exists i, j : \mathbb{N}.(f(i) = f(j) \ \&\ (i \neq j)).$$

[1]Chet Murthy became a key member of the Coq team after receiving his PhD from the Cornell CS Department. When he was a student, he built more than one version of Nuprl so that he could experiment by changing it. He also connected five versions of Nuprl into a networked prover that allowed him to explore topics for his PhD thesis. We called it "chet net." He was part of the INRIA team that won the ACM system award for Coq version 8.

One proof of this is to first assert that either there are infinitely many zeros in the range of f or infinitely many ones. This is justified by simple propositional reasoning, the law of excluded middle. In the zeros case, pick two zeros and in the ones case pick two ones. This proof is highly *non-constructive* and seems to invoke superhuman powers, to know the disjunction which we could describe as (Inf(0) or Inf(1)).

There is a much simpler constructive proof. We just examine f(0), f(1), f(2). There are only two choices for three slots, so two must be the same, and we can decide by evaluating at these three points if the function is computable. Even if f is not computable, we just mention the three values, and enumerate all possible choices of values. If the function is computable, we can easily decide these cases. But the conclusion does not require choices and does not build values.

What is remarkable for questions about natural numbers is that there is a theorem of logic that for propositions of the above form, "for all there exists," we can convert any classical proof in Peano Arithmetic, to a constructive one in Heyting Arithmetic. Harvey Friedman found a very simple proof of this deep fact [11], and Chetan Murthy implemented that method in Nuprl [16] so that Nuprl could actually carry out the translation. Chet then applied the transformation to an important theorem in classical number theory to solve an open problem [15].

2 The nature and future of proof assistants

We will use the Nuprl proof assistant [9, 2] as an example of the species of proof assistants. It is one of the oldest systems still in use, one that is steadily evolving, sometimes in in bursts of new ideas and enhancements. We will briefly consider the general nature of proof assistants and their future.

One of the first issues we face in "doing mathematics" with proof assistants is coping with the variety of interesting *notations and symbols* such as $\forall$, $\exists$, $\Rightarrow$, Σ, ∞, $\emptyset$, λ, ϵ, ω, $\cap$, $\cup$, $\aleph$, $\perp$, and so forth. In Nuprl this is done by having *display forms* that can use special characters. These are controlled at the level of "display" of the items stored in the Library of mathematical definitions and theorems.

Another major issue is deciding how to structure proofs and display them. In this realm a great deal was known from early work of logicians. We have seen the *tableau style* of proofs, and we mentioned *natural deduction.* You know that Nuprl created a style we call *refinement proofs* [6], closely related to tableau. One of the issues is whether to allow *derived rules.* Nuprl does not use them and MetaPRL did. We have not studied this interesting topic in the course. An issue we did investigate using Nuprl is the translation of refinement proofs into natural language. We studied natural language presentation of proofs with Cornell professors Cardie and Lee [13].

The issue of automating the construction of Nuprl proofs is based on the pioneering work of Robin Milner in his Edinburgh LCF system [12] and the notion of *proof tactics*. This topic is probably worth an entire course. The tactic mechanism is also used in Agda, Coq, HOL, Isabelle HOL, MetaPRL, and other systems. It was a "game changing" advance in the subject of automated reasoning. The LCF tactic system was implemented in Lisp.

The constructive proof assistants also include a programming language integral to their logic. This is typically an applied lambda calculus. In Coq the programming language is OCaml. In the Trellys project at UPenn it is Haskell. At Microsoft they could use F sharp. Nuprl has its own rich functional programming language defined along with the logic. It includes some novel features and is about to add another fundamental notion, the concept of a *free choice sequence* [19, 20] from Brouwer's intuitionistic mathematics.

The proof assistants organize their results into *libraries of formal mathematics*. The Nuprl library is called the FDL (Formal Digital Library) [3], and it also contains some results from other proof assistants [4]. Results in these libraries are sometimes connected to specific books and other resources. A good example is the connection of the definitions and theorems of chapter two of the book *Constructive Analysis* [8] to their formalization in Nuprl. This resource was created by Mark Bickford and Richard Eaton.

We see that proof assistants are already advancing science and mathematics by producing the most reliable mathematical results known. They are also being used to build reliable software systems and to defend the computing infrastructure from cyber attack by creating *attack tolerant systems* [17, 18]. Systems built using Nuprl have been deployed in industry and elsewhere, e.g. NASA and DoD [14, 1, 7, 21]. An interesting popular account of this topic entitled *Beyond Knowledge* was written by J. Aron for *New Scientist* [5].

References

[1] Mark Aagaard and Miriam Leeser. Verifying a logic synthesis tool in Nuprl. In Gregor Bochmann and David Probst, editors, *Proceedings of Workshop on Computer-Aided Verification*, pages 72–83. Springer-Verlag, June 1992.

[2] Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. *Journal of Applied Logic*, 4(4):428–469, 2006.

[3] Stuart Allen, Mark Bickford, Robert Constable, et al. FDL: A prototype formal digital library. PostScript document on website, May 2002. `http://www.nuprl.org/html/FDLProject/02cucs-fdl.html`.

[4] Stuart F. Allen, Mark Bickford, Robert Constable, Richard Eaton, and Christoph Kreitz. A Nuprl–PVS connection: Integrating libraries of formal mathematics. Technical Report TR2003-1889, Cornell University, Ithaca, New York, 2003.

[5] Jacob Aron. Beyond Knowledge. *New Scientist*, pages 28 – 31, 2015.

[6] J. L. Bates. *A Logic for Correct Program Development.* PhD thesis, Cornell University, 1979.

[7] Ken Birman, Robert Constable, Mark Hayden, Jason Hickey, Christoph Kreitz, Robbert van Renesse, Ohad Rodeh, and Werner Vogels. The Horus and Ensemble projects: Accomplishments and limitations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, pages 149–161, Hilton Head, SC, 2000. IEEE Computer Society Press.

[8] E. Bishop and D. Bridges. *Constructive Analysis.* Springer, New York, 1985.

[9] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, NJ, 1986.

[10] Robert L. Constable and Chetan Murthy. Finding computational content from classical proofs. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 341–362. Cambridge University Press, 1991.

[11] Harvey Friedman. Classically and intuitionistically provably recursive functions. In D. S. Scott and G. H. Muller, editors, *Higher Set Theory*, volume 699 of *Lecture Notes in Mathematics*, pages 21–28. Springer-Verlag, 1978.

[12] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: a mechanized logic of computation*, volume 78 of *Lecture Notes in Computer Science.* Springer-Verlag, NY, 1979.

[13] Amanda Holland-Minkley, Regina Barzilay, and Robert L. Constable. Verbalization of high-level formal proofs. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 277–284. AAAI, July 1999.

[14] Miriam Leeser. Using Nuprl for the verification and synthesis of hardware. *Phil. Trans. Royal Society of London*, 339:49–68, 1992.

[15] Chetan Murthy. *Extracting Constructive Content from Classical Proofs.* PhD thesis, Cornell University, Department of Computer Science, 1990. (TR 90-1151).

[16] Chetan Murthy. An evaluation semantics for classical proofs. In *Proceedings of the 6th Symposium on Logic in Computer Science*, pages 96–109, Vrije University, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.

[17] Vincent Rahli, Nicolas Schiper, Robbert Van Renesse, Mark Bickford, and Robert L. Constable. A diversified and correct-by-construction broadcast service. In *The 2nd International Workshop on Rigorous Protocol Engineering (WRiPE)*, Austin, TX, October 2012.

[18] Nicolas Schiper, Vincent Rahli, Robbert Van Renesse, Mark Bickford, and Robert L. Constable. Developing correctly replicated databases using formal tools, 2014. Accepted to DSN 2014.

[19] A.S. Troelstra. *Choice Sequences.* Oxford University Press, Oxford, 1977.

[20] Mark van Atten. *On Brouwer.* Wadsworth Philosophers Series. Thompson/Wadsworth, Toronto, Canada, 2004.

[21] Robbert van Renesse, Kenneth P. Birman, Mark Hayden, Alexey Vaysburg, and David Karr. Building adaptive systems using Ensemble. *Software: Practice and Experience*, 28(9):963–979, July 1998.