CS/Math-4860 Course Description for Fall 2018

Robert L. Constable

August 6, 2018

Abstract

Applied logic is increasingly important in mathematics, philosophy, and computer science. Cornell recognized this trend early and created the course Applied Logic, joint between Computer Science and Mathematics.

This document provides an overview of the main topics in the Fall 2018 version of the course taught by Computer Science. The course will stress the relevance of the topics to computer science, mathematics, and philosophy. More information will be available at the CS course web page.

1 Relevance of Logic

The importance of logic to mathematics and philosophy is clear. Mathematicians develop a subject by formulating and proving theorems about it. The idea of *proof* goes back at least to Euclid and Aristotle. The notion of a *constructive proof* arose in Euclidean geometry because many proofs provided ruler and compass constructions. Aristotle classified proofs based on their structure. Many of his ideas remain relevant.

The importance of logic for computer science was well established at least by 1969 in the article by C.A.R.Hoare, *An axiomatic basis for computer programming*, published in the *Communications of the ACM* [13]. This major accomplishment and the large body of subsequent work earned Tony Hoare a knighthood. His idea is that the task to be achieved by a program should be specified precisely in a logical language, and then using his proposed new axioms, it would be possible to prove that a program accomplished the task for which it was written.

By the 1980's there were implemented formal systems that integrated programs, assertions, and proofs that the program accomplished the task defined by its formal logical specification. The language of specification was not the language of sets common in mathematics. The *programming languages used types instead of sets.* At Cornell we produced the PLCV system based on the Cornell designed programming language PLCV [8]. This led to the idea of *proofs as programs* [2] and helped advance the use of types rather than sets as the foundational concept. The idea of types came from the famous work of Whitehead and Russell, *Principia Mathematica* [20] that provided a foundation for mathematics based on types rather than sets. This theory was later simplified considerably by Church [5]. There is an interesting comic book account by A. Doxiadis and C. Papadimitriou, *Logicomix: An Epic Search for Truth* [10].

A major addition to applied logic was the creation of software systems called *proof assistants* or *provers.* These systems help users create formal proofs in type theory. They greatly expand the role of formal proofs. The PLCV system used a simple type theory, but *the new proof assistants aimed to support a type theory capable of expressing*

all of mathematics with a special stress on computational mathematics. The *Nuprl* proof assistant built at Cornell was released in 1986 in the book *Implementing Mathematics* [7]. The *Coq* proof assistant was introduced in 1988 by T. Coquand and G. Huet in the article *The calculus of constructions* [9]. Both have been operational and evolving ever since those times. They have already been used to produce significant software systems known to be formally correct and to prove new results in mathematics and computing theory [1].

Open problems in mathematics have been solved by these proof assistants [12, 16, 14, 18, 19, 6]. The Coq system is supported by the French research organization at INRIA, and Nuprl remains at Cornell University supported by US research funds from the NSF and the DoD (e.g. DARPA, AFRL, and ONR).

Proof assistants help users prove theorems in mathematics and computer science, but are perhaps best known for verifying software, i.e. programs, protocols, compilers, operating system kernels, and so forth. Investments in proof assistants are making them "smarter" in the sense that they can guide users more quickly and surely toward an answer and check that all the reasoning steps are correct. The more scientists use these tools, the more facts and methods are formalized by proof assistants. Usually these have inscrutable names such as Agda [4], Coq [3], HOL [11], Lean, Nuprl [7], PVS [15] and so forth, not Alice and Bob. So let us name one of the hypothetically most potent proof assistants *Quin*, **Q** for short.

The jargon is that expert users "drive" a proof assistant. As they drive it, knowledge is archived and proof methods and "tactics" are saved so they can be applied automatically in other situations. In some cases the tactics allow **Q** to finish solving the problem before the user realizes that it has a chance. The time thus saved is a strong motivator to let the proof assistant guide users along a particular path that the tactics determine, even if there is a simpler way to solve the problem.

The situation we now face is that people who use proof assistants come to depend on them more and more, as we do with most thought tools, like spell checkers. Should there be a counter force? Perhaps suggestions from **Q** at certain points that there might be a better step that a human could discover but **Q** does not yet know. Perhaps a *feedback mechanism* where readers grade proofs by their clarity, cleverness, and elegance. Those grades might help proof assistants do a better job – not because of the proof assistant's enjoyment and happiness but because of the user's.

In due course, the role of proof assistants will expand as we learn to incorporate more and more elements of *Artificial Intelligence*(AI) into their operation. This course will speculate on such applications and look at research questions at the intersection of AI and Applied Logic [17].

References

[1] 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.

[2] J. L. Bates and Robert L. Constable. Proofs as programs. *ACM Transactions of Programming Language Systems*, 7(1):53–71, 1985.

[3] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development; Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer-Verlag, 2004.

[4] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of agda – a functional language with dependent types. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *LNCS 5674, Theorem Proving in Higher Order Logics*, pages 73–78. Springer, 2009.

[5] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:55–68, 1940.

[6] Robert Constable and Mark Bickford. Intuitionistic Completeness of First-Order Logic. *Annals of Pure and Applied Logic*, 165(1):164–198, January 2014.

[7] 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.

[8] Robert L. Constable, S. Johnson, and C. Eichenlaub. *Introduction to the PL/CV2 Programming Logic*, volume 135 of *Lecture Notes in Computer Science*. Springer-Verlag, NY, 1982.

[9] Thierry Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[10] A Doxiadis and C. Papadimitriou. *Logicomix: An Epic Search for Truth*. Ikaros Publications, Greece, 2008.

[11] Michael Gordon and Tom Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, Cambridge, 1993.

[12] T. Griffin. A formulas-as-types notion of control. In *Proceedings of the Seventeeth Annual Symposium on Principles of Programming Languages*, pages 47–58, 1990.

[13] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580,583, 1969.

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

[15] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 411–14. Springer-Verlag, New Brunswick NJ, July–August 1996.

[16] J. Russell and Chetan Murthy. A direct constructive proof of Higman's Lemma. In *Proceedings of the 5^{th} Symposium on Logic in Computer Science*, pages 257–269. IEEE Computer Society Press, June 1990.

[17] S. Schmitt, L. Lorigo, C. Kreitz, and A. Nogin. JProver: Integrating connection-based theorem proving into interactive proof assistants. In *IJCAR, LNAI 2083*, pages 421–426. Springer, 2001.

[18] Judith L. Underwood. A constructive completeness proof for the intuitionistic propositional calculus. Technical Report 90–1179, Cornell University, 1990.

[19] Judith L. Underwood. The tableau algorithm for intuitionistic propositional calculus as a constructive completeness proof. In *Proceedings of the Workshop on Theorem Proving with Analytic Tableaux, Marseille, France*, pages 245–248, 1993. Available as Technical Report MPI-I-93-213 Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[20] A.N. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 2nd edition, 1925–27.