Applied Logic Fall 2018 Final Lecture

December 4, 2018

Abstract

This offering of Applied Logic has used type theory to unify and relate the key concepts in logic, starting with the Propositional Calculus both classical (PC) and intuitionistic (iPC). Following that subject we studied First-Order Logic, both classical (FOL) [28] and intuitionistic (iFOL) [10]. We noted that for the standard classical foundation of mathematics, Zermelo/Fraenkel Set Theory with the Axiom of Choice (ZFC), FOL is sufficient.

We noted that *type theory* connected these fundamental ideas in mathematics with basic notions needed for the foundations of computer science. One of the important ideas not often covered in this course is Judith Underwood's result [29, 30] that the decision procedure for iPC allows us to know when a specification is not "programmable." We studied this important result in Lecture 15 after the class was familiar with and comfortable with types.

Already in Lecture 2 we informally introduced the idea that type theory is a good foundation for computer science, in part because programming languages use types to to classify data and specify programming tasks. This was major theme in the course right from the start. We introduced early on in Lecture 5 the notion of *evidence semantics* to explain the role of types in logic. We illustrated evidence using *Euclidean Geometry* [20]. Here we also introduced the notion of *synthetic algorithms* in contrast to numerical algorithms.

By Lecture 22 we looked at how C.A.R. Hoare [15, 16, 17] and John McCarthy [23, 24] used types to describe programming tasks. We discussed the influence of *Principia Mathematica* [31] in the development of modern type theory. We briefly mentioned the effort to create a programming language, SETL [25] based on set theory rather than type theory. The SETL effort did not attract much support.

In Lectures 24 and 25 we looked at other basic research articles that helped establish type theory in computer science, in particular the work of McCarthy [24] and Scott [26, 27] and the article of Martin-Löf [22] in which he discovered the connections of type theory to computer science and mentioned some early articles. He did not pursue the connection in detail himself. On the other hand, he had a large impact on computer science in Sweden, and the abiding interest there has led to the long term development of the proof assistant *Agda* [4]. The French were also drawn into type theory based on fundamental results of the logician Girard [11]. They built the first version of the Coq systems based on this work. When my good friend Gilles Kahn [18, 19] merged the Coq project with the OCaml project, Coq took its modern shape which is much closer to Nuprl. In this lecture we look forward at the on-going development of type theory and its role in both mathematics and computer science. We will say a bit more about the value of *dual-prover-technology*, as in Coq/Nuprl, where the Coq proof assistant is used to prove the correctness of Nuprl's rules.

1 Type Theory and Proof Assistants of the Future

Over the last fifty years there has been a steady increase in the effectiveness of *proof assistants*. We see in them the increasing importance of automated reasoning in solving open problems in mathematics and computer science. Proof assistants are also used to build highly reliable software with provable properties. Active researchers in this area predict advances that will be regarded as *game changing*. We expect that the US and EU will continue the highly productive cooperation, dating back to de Bruijn's *Automath* [9], to the creation and deployment of modern proof assistants, such as *Agda* [4], *Coq* [7, 8, 2], and HOL [12, 14], and Nuprl [6] among others.

Below we briefly enumerate some *new mechanisms* which we believe will significantly improve the reasoning capabilities of proof assistants. We have been experimenting with these new reasoning mechanisms using *Nuprl*. The results are promising. These mechanisms will significantly increase the value of proof assistants. Our future research will validate this claim by implementing, applying, and demonstrating the new proposed mechanisms. We have recently enriched the type theory implemented by Nuprl, this provides the richest logical foundation known for testing the proposed new reasoning mechanisms.

The PRL research group also intends to explore the use Nuprl in giving a rigorous formal account of concepts in quantum physics, inspired by the article *Constructive Mathematics and Quantum Physics* [5]. We have an opportunity here since one of the PRL group researchers whom you have met, Ariel Kellison, has a degree in physics.

1. The PRL Group at Cornell is one of the few research groups that extensively uses two proof assistants, Coq and Nuprl. We use Coq to prove the correctness of Nuprl rules and explore extensions to those rules. This has allowed us to implement strong criteria for the foundational value of proof assistants for which Vladimir Voevodsky had also advocated. There is very interesting research required to enhance the reasoning power of the Coq/Nuprl combination and **extend the capabilities and reach of modern proof assistants** [3]. If there is time in the final lecture, we will look at how Dr. Bickford recently changed the definition of the Nuprl equality type (x = y in T).

   A consequence of the definition of the equality type is that (t = t' in T) is always a type in context H if T is a type in context H and t and t' are closed terms (so they are in Base). Thus (t in T) is always a type for T a type and t a *closed term*.

2. **Tactics, decision procedures, and a collaboration infrastructure** are central to the automated reasoning mechanisms of the *Nuprl* proof assistant. The **Formal Digital Library**, **FDL**, is a key unique resource for this proof assistant [21]. The FDL is the dynamic repository for all formal definitions, theorems, proofs, and tactics. It enables communication among users and the proof assistant who coordinate on solving problems and proving theorems. We plan to deploy these new AI techniques to help us improve the effectiveness of both Nuprl proofs and the Coq proofs used to augment and support the Nuprl implementation.

3. **Adding transformations**. This role for transformations came from thinking about how to go beyond the 504 tactics [13, 1], tacticals, and transformation scripts we now use to assist in proof construction. One transformation that is especially clear and useful arises from thinking about how to extend results from Real Analysis to Complex Analysis. It is in part inspired by a quote from Bishop and Bridges that suggests a high level script for how the basic arithmetical results of complex analysis could be algorithmically constructed using the extensive formalization of real analysis as a *template*. We believe that there are effective ways to "implement" these templates. This investigation led us to thinking about reasoning mechanisms that go beyond tactics and yet can be implemented.

4. Having data about Nuprl theories might help explain the new formal techniques we plan to explore. For the library of *constructive real analysis* Mark collected some data: 272 definitions, 1,731 objects. In Reals-2

there are 80 definitions and 449 lemmas. Having data about our library and results might be a valuable resource.

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] 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.

[3] Mark Bickford, Liron Cohen, Robert Constable, and Vincent Rahli. Computability beyond Church-Turing via Choice Sequences. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 266 – 279, 2018.

[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] Douglas Bridges and Karl Svoil. Constructive mathematics and quantum physics. *International Journal of Physics*, 39:503 – 515, 2000.

[6] 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.

[7] Thierry Coquand. Metamathematical investigations of a calculus of constructions. In P. Odifreddi, editor, *Logic and Computer Science*, pages 91–122. Academic Press, London, 1990.

[8] Cristina Cornes, Judicaël Courant, Jean-Christophe Filliâtre, Gérard Huet, Pascal Manoury, Christine Paulin-Mohring, César Muñoz, Chetan Murthy, Catherine Parent, Amokrane Saïbi, and Benjamin Werner. The Coq Proof Assistant reference manual. Technical report, INRIA, 1995.

[9] N. G. de Bruijn. The mathematical language Automath: its usage and some of its extensions. In J. P. Seldin and J. R. Hindley, editors, *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61. Springer-Verlag, 1970.

[10] M. Fitting. *Intuitionistic model theory and forcing*. North-Holland, Amsterdam, 1969.

[11] Jean-Yves Girard. *Interprétation Fonctionelle et Élimination des Compures de l'Arithmétic d'Ordre Supérieur*. PhD thesis, Université Paris VII, 1972.

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

[13] 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.

[14] John Harrison. HOLLight: A tutorial introduction. In *Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer, 1996.

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

[16] C. A. R. Hoare. Notes on data structuring. In *Structured Programming*. Academic Press, New York, 1972.

[17] C. A. R. Hoare. Recursive data structures. *International Comput. Inform. Sci.*, 4(2):105–32, June 1975.

[18] Gilles Kahn. Natural semantics. In G. Vidal-Naquet F. Brandenburg and M. Wirsing, editors, *STACS '87*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, Berlin, 1987.

[19] Gilles Kahn. Constructive geometry according to jan von plato. V5.10, 1995.

[20] Ariel Kellison, Mark Bickford, and Robert Constable. Implementing Euclid's straightedge and compass constructions in type theory. *Annals of Mathematics and Artificial Intelligence*, Sep 2018.

[21] Lori Lorigo. *Information Management in the Service of Knowledge and Discovery*. PhD thesis, Cornell University, 2006.

[22] Per Martin-Löf. Constructive mathematics and computer programming. In *Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North Holland.

[23] J. McCarthy. Computer programs for checking mathematical proofs. In *Proceedings of the Symposium in Pure Math, Recursive Function Theory*, volume V, pages 219–228. AMS, Providence, RI, 1962.

[24] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland, Amsterdam, 1963.

[25] E. Schonberg, J. Schwartz, and M. Sharir. An automatic technique for the selection of data represntations in SETL programs. *ACM Transactions of Programming Language Systems*, 3(2):126–143, April 1981.

[26] D. Scott. Constructive validity. In D. Lacombe M. Laudelt, editor, *Symposium on Automatic Demonstration*, volume 5(3) of *Lecture Notes in Mathematics*, pages 237–275. Springer-Verlag, NY, 1970.

[27] D. Scott. Logic and programming languages. *Comm. Assoc. Comput. Mach.*, 20:634–641, 1977.

[28] R. M. Smullyan. *First–Order Logic*. Springer-Verlag, New York, 1968.

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

[30] 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.

[31] A.N. Whitehead and B. Russell. *Principia Mathematica*, volume 1, 2, 3. Cambridge University Press, 2nd edition, 1925–27.