CS6180: Lecture 3

Robert Constable

August 28, 2017

1 Lecture Summary

First we will finish the brief account of the history and motivations for the creation of constructive type theory and its supporting proof assistants. Then we will look closely at Kleene's axioms [16] for *Peano Arithmetic* (PA) and its constructive counter part *Heyting Arithmetic* (HA). These are presented in the formal logical language called *first-order logic*, the logical system that arose from Frege's ground breaking work, *Begriffsschrift* [9] mentioned in Lecture 2. This is the same logical language used to formalize Zermelo-Fraenkel set theory with the axiom of choice, *ZFC* [12, 8, 7]. So in a sense, many mathematicians and students believe that this is the only logical language they need to know. All of modern mathematics can be axiomatized in this logic.[1]

We will not only look at a formal axiomatization for PA and HA, we will also discuss the various styles in which proofs in first order logic are organized. Kleene starts with what is called a *Hilbert style axiomatization*. We will also discuss the *natural deduction style* [20], and the *sequent style* of Gentzen [10]. Nuprl uses a top down version of sequents which we called *refinement style deduction*. It fits particularly well with the LCF approach to automated deduction pioneered by Milner [11, 18], *Edinburgh LCF: a mechanized logic of computation*. Another popular style of deduction are the *tableaux systems* used by Smullyan [23] used in his outstanding logic book *First-Order Logic*.

We will also finish up the brief history of logic by pointing out why Russell and Frege were searching for a rigorous logical foundation for mathematics at the same time. It was their response to a certain "crisis in mathematics" in roughly the 1870's. The crisis arose during efforts to define continuous functions and understand the convergence of infinite series. We will post some rough notes on this topic in the *Story of Logic* written for CS4860.

Our integration of a programming language and its programming logic into a single system is one origin of proof assistants for constructive type theory. This integration injects assertions as *dependent types* into a programming language.[2] Programming logics remain a viable notion, especially for modern programming languages like OCaml, Haskell, Scala, etc. The term "proof assistant" was not standard when we wrote the book about the *PLCV Programming Logic* [4, 6].

[1] One of the first logical systems used for automated reasoning was called FOL [25].

[2] The standard account of dependent types does not mention the PLCV work because that system was built before the idea had its current name.

Indeed, we integrated the logic with the programming language. Even though it was possible to simply write and check formal proofs in PLCV and we did use the system in that way as well. PLCV was mainly designed to assist in creating verified programs. We said things like this is an example of a formal programming logic. This kind of system would be viable again for a very rich functional programming language with dependent types. In due course it might be the case that all high quality programming languages come with a fully integrated logic, and the term proof assistant will disappear. We'll talk about *programming assistants* or logical program development environments, *Logical-PDE*.

2 First-Order Logic and Peano Arithmetic

We will discuss various formats for presenting formal proofs. The most basic is called the *Hilbert style* [14] due to David Hilbert. Proofs in this style are simply finite lists of *axioms* or applications for *inference rules* to previously listed formulas. We give Kleene's axioms for arithmetic in this style from his book *Introduction to Metamathematics* [16]. He covers both the classical theory of *Peano Arithmetic* (PA) [19] and the constructive theory of *Heyting Arithmetic* [13]. Proofs in this style are not easy to read, but their definition is simple.

Another style is *natural deduction* [21, 20] due to due to Dag Prawitz. The PLCV proofs we show are basically in this style. It reads somewhat like the proofs we see in textbooks. Another important formalism for proofs is due to Gerhard Gentzen [10] called *sequent style*. Gentzen proved very interesting results about the normal forms of such proofs. The tableaux proof style was invented by K.J.J.Hintikka [15] and used with great clarity by Smullyan in his classic textbook *First-Order Logic* [22]. We will probably discuss this system as well in due course. The Nuprl system uses yet another formalism create called *refinement proofs* used in Nuprl based on work of J. Bates [1, 5]. This is a top down version of Gentzen's sequent style. Nuprl also uses the *LCF tactic mechanism* [11] to automate proof construction.

We will start with Kleene's account of first order logic and Peano Arithmetic [16] on page 82 and recommend a simple presentation of first order logic by Smullyan [23] as optional supplementary reading that might be mentioned from time to time. Here we list Kleene's axioms and inference rules. Then we discuss how these are used in a formal system to build theories. If these logical formulas are totally mysterious, then it is best to take the course CS4860 (Math4860) on *Applied Logic* which CS will offer in the spring semester.

Below are the axioms from Kleene's book cited above. They are listed here to illustrate formal logic. The course assumes some familiarity with logical notation but not necessarily with formal theories. So most students should find these axioms readable, but not that many will know how to create a useable formal theory from them. We will explore that issue already in the first two lectures.

Propositional Calculus Axioms

1a. $A \Rightarrow (B \Rightarrow A)$.

1b. $(A \Rightarrow B) \Rightarrow ((A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow C))$.

Inference Rule 2:

$$\frac{A, A \Rightarrow B}{B}$$

3. $A \Rightarrow (B \Rightarrow A\&B)$.
4a. $(A\&B) \Rightarrow A$.
4b. $(A\&B) \Rightarrow B$.
5a. $A \Rightarrow (A \vee B)$.
5b. $B \Rightarrow (A \vee B)$.
6. $(A \Rightarrow C) \Rightarrow ((B \Rightarrow C) \Rightarrow ((A \vee B) \Rightarrow C))$.
7. $(A \Rightarrow B) \Rightarrow ((A \Rightarrow \sim B) \Rightarrow \sim A)$.
8*. $\sim\sim A \Rightarrow A$. **classical**

Predicate Calculus Axioms

Inference Rule9:

$$\frac{C \Rightarrow A(x)}{C \Rightarrow \forall x.A(x)}$$

10. $\forall x.A(x) \Rightarrow A(t)$.
11. $A(t) \Rightarrow \exists x.A(x)$.

Inference Rule 12:

$$\frac{A(x) \Rightarrow C}{\exists x.A(x) \Rightarrow C}$$

Number Theory Axioms

13. $A(0)\&\forall x.(A(x) \Rightarrow A(x')) \Rightarrow A(x)$.
14. $a' = b' \Rightarrow a = b$.
15. $\sim (a' = 0)$.
16. $a = b \Rightarrow (a = c) \Rightarrow b = c$.

17. $a = b \Rightarrow a' = b'$.
18. $a + 0 = a$.
19. $a + b' = (a + b)'$.
20. $a \times 0 = 0$.
21. $a \times b' = (a \times b) + a$.

How do we use these axioms to create proofs? Kleene gives this example on page 85. This is an example of what logicians call a Hilbert style axiom system in which there is only one proof rule in addition to the axioms. The is the rule of *modus ponens*, if we know A and $A \Rightarrow B$, then we can deduce B.

1.$A \Rightarrow (A \Rightarrow A)$. Axiom schema 1a.
2.$(A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow ((A \Rightarrow A) \Rightarrow A)) \Rightarrow (A \Rightarrow A)$. Schema 1b.
3.$(A \Rightarrow ((A \Rightarrow A) \Rightarrow A))$. Rule 2,1,2
4. $A \Rightarrow ((A \Rightarrow A) \Rightarrow A)$ Axiom schema 1a.
5.$(A \Rightarrow A)$. Rule 2,4,3.

Kleene also shows on page 84 how to prove $a = a$ for numerical variables a in 17 lines of axioms and inference rules. This kind of proof makes formal logic and formal mathematics seem *impossibly tedious* and essentially incomprehensible. What computer science brought to this research early on is the notion that machines could conduct these tedious proofs in the background and achieve an exceptionally strong degree of formal correctness. In their classic paper, *Empirical Explorations with the Logic Theory Machine: A case study in heuristics* [17], Newell, Shaw, and Simon demonstrated that computers could execute the tediously long proofs and check all the details so that humans would not be required to function at these low levels. This was a revolutionary advance in automated reasoning that inspired decades of further research to create the AI tools that make proof assistants possible.

We will examine an example from Kleene showing how to structure proofs as trees using Frege's turnstile symbol, $\vdash$, separating hypotheses from the goal to be proved, $H \vdash G$ where the hypotheses H is a *list* of labled formulas and variable declarations, and the goal is a single formula. Such expressions are called *sequents*. In the Nuprl book they are written as $H >> G$. In these notes we use $x_1 : A_1, ..., x_n : A_n \vdash G$. Some formalisms, such as tableaux, allow multiple goals, say $G_1, ..., G_m$, but Nuprl does not.

3 Comments on the PLCV Programming Logic

The supplementary material for this lecture includes an example of an *asserted program* to find the least prime factor of a natural number. It is needed in a program implementing the *fundamental theorem of arithmetic* that every natural number is the product of a unique sequence of prime numbers. This proof is constructive in the sense that the program actually finds the unique prime

decomposition of a number. This somewhat dense program/proof combination provides both an imperative program to find the factors combined with a formal proof that the algorithm is correct. This is an easy concept to grasp and does not require an extensive discussion of constructive logic and constructive mathematics. It is just "asserted programming." College freshmen can easily grasp this idea.

This very simple technology introduces a deep idea in logic. It shows how programs and proofs can be combined to the enrichment of both constructs. This was a dream that the author dreamt in the 1971 article "Constructive Mathematics and Automatic Program Writers" [2]. It launched the research program that led to PLCV, Nuprl, and its constructive type theory created by the author, Joe Bates, and the large team of graduate students who helped write the book *Implementing Mathematics* [3] and the even larger number who have continued the development of the type theory and its proof assistant. We have kept extending the theory and the system year by year ever since 1984. In the last two years the theory has become a full fledged *intuitionistic type theory*. That we have never taught before this course, and no other proof assistant implements such a rich foundational theory of computing.

References

[1] J. L. Bates. A logic for correct program development. Technical Report TR81-455, Cornell University, 1981. Revision of Cornell University Ph.D. thesis submitted August 1979.

[2] Robert L. Constable. Constructive mathematics and automatic program writers. In *Proceedings of the IFIP Congress*, pages 229–233. North-Holland, 1971.

[3] 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.

[4] 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.

[5] Robert L. Constable, T. Knoblock, and J. L. Bates. Writing programs that construct proofs. *Journal of Automated Reasoning*, 1(3):285–326, 1984.

[6] Robert L. Constable and Michael J. O'Donnell. *A Programming Logic*. Winthrop, Mass., 1978.

[7] A. A. Fraenkel and Y. Bar Hillel. *Foundations of Set Theory*, volume 67 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 2nd edition, 1958.

[8] A. A. Fraenkel, Y. Bar-Hillel, and A. Levy. *Foundations of Set Theory*, volume 67 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 2nd edition, 1984.

[9] Gottlob Frege. Begriffsschrift, a formula language, modeled upon that for arithmetic for pure thought. In van Heijenoort [24], pages 1–82.

[10] Gerhard Gentzen. Investigations into logical deduction (1934). In M. Szalo, editor, *The Collected Paers of Gerhard Gentzen*. North-Holland, Amsterdam, 1969.

[11] 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.

[12] Paul R. Halmos. *Naive Set Theory*. Springer-Verlag, New York, 1974.

[13] A. Heyting. *Intuitionism, An Introduction*. North-Holland, Amsterdam, 1966.

[14] David Hilbert. The foundations of mathematics. In van Heijenoort [24], pages 464–479. reprint of 1928 paper.

[15] K. J. J. Hintikka. Form and content in quantification theory. *Acta Philosophica Fennica*, 8:7 – 55, 1955.

[16] S. C. Kleene. *Introduction to Metamathematics.* D. Van Nostrand, Princeton, 1952.

[17] A. Newell, J.C. Shaw, and H.A. Simon. Empirical explorations with the logic theory machine: A case study in heuristics. In *Proceedings West Joint Computer Conference*, pages 218–239, 1957.

[18] L. Paulson. Lessons learned from LCF: a survey of natural deduction proofs. *Comp. J.*, 28(5):474–479, 1985.

[19] G. Peano. *Selected Works*. University of Toronto Press, Toronto, 1973. Edited by H.C. Kennedy.

[20] D. Prawitz. *Natural Deduction*. Dover Publications, New York, 1965.

[21] D. Prawitz. *Natural Deduction*. Almquist and Wiksell, Stockholm, 1965.

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

[23] Raymond M. Smullyan. *First-Order Logic*. Dover Publications, New York, 1995.

[24] J. van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, MA, 1967.

[25] Weyhrauch and C. L. Talcott. The logic of fol systems: Formulated in set theory. In M. Hagiya, N. D. Jones, and M. Sato, editors, *Festschrift in Honor of Professor Satoru Takasu*, number 792 in Lecture Notes in Computer Science. Springer-Verlag, 1994.