CS6180: Lecture 2

Robert Constable

August 29, 2017

1 Lecture Summary

The supplemental material for this lecture includes a proof of the Fundamental Theorem of Arithmetic in the PLCV system. It is presented as an example of the integration of a programming language and its logic. This approach to documenting programs and proving them correct is well illustrated by the recent Dafny system [12]. We talked briefly about the PLCV proof assistant in the first lecture. One reason for doing that is to trace the use of the term "type" as it is used in the phrase "constructive type theory." Another point was to connect programming and type theory and the idea of asserted-programs-as-proofs or just *programs as proofs* for short. The notion of *proofs as programs*[1] is a bit more mysterious and compelling, and we will look at that in detail several lectures hence.

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.[1] 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, 5]. 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*.

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

2 Importance of Type Theory in Mathematics and Computer Science

Types played a very important and prominent role in providing a foundation for mathematics, going back to the 1925 monumental three volume work by Alfred North Whitehead and Bertrand Russell, *Principia Mathematica* [22] (also featured in Logicomic cited above.) The course material includes the appendix to Russell's book *The Principles of Mathematics* [20]. This is the origin of type theory that was expanded to the very dense and heavily symbolic and voluminous three volume book by Whitehead and Russell *Principia Mathematica* [22].[2]

Type theory was simplified by Church [2] in his *Simple Theory of Types,* and types gradually appeared in computer science via programming languages such as Algol 60 and FORTRAN and later Pascal [9] and Forsythe [19]. In these languages, types are used to distinguish different kinds of data, e.g. fixed point numbers, floating points numbers, arrays and so forth. As programming languages became more expressive, their type systems were steadily enriched. Hoare and Reynolds wrote very influential articles that stressed the importance of types [7, 9, 8, 14, 17, 18]. As it became more and more important to reason carefully about whether programs accomplished the tasks they were intended to accomplish, the importance of types increased further. Languages such as Algol 68 had quite rich type systems. John Reynolds wrote extensively on this topic and designed the language Forsythe [19]. He encouraged the further enrichment of types systems for programming [15, 16, 18]. The programming language Pascal [9, 10] played a major role in showing the advantages of a rich type system for explaining programming tasks.

3 First-Order Logic and Peano Arithmetic a la Kleene

At the same time that Russell was working to provide a firm foundation for mathematics, the German logician Gottlob Frege was writing his approach to this same task, *Begriffsschrift, A Formula Language, modeled Upon that for Arithmetic for Pure Thought* [6] which was the basis for his foundational work. Russell wrote to Frege about the paradox he had found in his efforts to use set theory as a foundation, the famous Russell paradox of the set of all sets that do not contain themselves. Frege realized that a similar paradox lurked in his foundational masterwork. He wrote a famous letter back to Russell and then attempted to salvage his system which he was unable to to. The same fate befell Peano who was writing a foundational work on arithmetic, but he was unphased by this because he owned a printing company. He was able to fix the problem with his less ambitious goal. We will look at his efforts next. Researchers have combined Peano's axioms with Frege's simple first-order logic from *Begriffsschrift.*

We will use Kleene's account of first order logic and Peano Arithmetic [11] on page 82 and recommend a simple presentation of first order logic by Smullyan [21] as optional supplementary

[2] Apparently Whitehead's name comes first because he was Russell's PhD supervisor. I learned this at the Principia Symposium [3].

reading that might be mentioned from time to time. Here we list the axioms. 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 I will teach in the spring semester.

Below are the axioms from Kleene's book. 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.$(A \Rightarrow ((A \Rightarrow A) \Rightarrow A))$. Axiom schema 1a.
4.$(A \Rightarrow A)$. Rules 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* [13], 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 separating hypotheses from the goal to be proved, $H \vdash G$ where the hypotheses H is a list of formulas and variable declarations and the goal is a single formula. These expressions are called *sequents*. In the Nuprl book they are written as $H >> G$. Some formalisms, such as

tableaux, allow multiple goals, but Nuprl does not.

References

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

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

[3] Robert Constable. The triumph of types: Principia mathematica's impact on computer science. In *Principia Mathematica anniversary symposium*, 2010.

[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 and Michael J. O'Donnell. *A Programming Logic*. Winthrop, Cambridge, Mass., 1978.

[6] Gottlob Frege. Begriffsschrift, a formula language, modeled upon that for arithmetic for pure thought. In J. van Heijenoort, editor, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, pages 1–82. Harvard University Press, Cambridge, MA, 1967.

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

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

[9] C. A. R. Hoare and N. Wirth. An axiomatic definition of the programming language Pascal. *Acta Informatica*, 2:333–335, 1973.

[10] K. Jensen and N. Wirth. *PASCAL user manual and report*. Springer-Verlag, New York, 1974.

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

[12] K. Rustan M. Leino. *Dafny: An Automatic Program Verifier for Functional Correctness*, pages 348–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[13] 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.

[14] John C. Reynolds. Towards a theory of type structure. In *Proceedings Colloque sur, la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–23. Springer-Verlag, New York, 1974.

[15] John C. Reynolds. The essence of Algol. In J. de Bakker and J. van Vliet, editors, *International Symposium on Algorithmic Languages*, pages 345–372. North-Holland, Amsterdam, 1981.

[16] John C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing '83*, pages 513–523. North-Holland, Amsterdam, 1983.

[17] John C. Reynolds. Polymorphism is not set-theoretic. In *Proceedings International Symposium on Sematics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 145–156, Berlin, 1984. Springer.

[18] John C. Reynolds. The coherence of languages with intersection types. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software, International Conference TACS '91*, volume 526 of *Lecture Notes in Computer Science*, pages 675–700. Springer-Verlag, Sendai, Japan, 1991.

[19] John C. Reynolds. Design of the programming language forsythe. Technical Report CMU-CS-96-146, Carnegie Mellon University, June 1996.

[20] Bertrand Russell. *The Principles of Mathematics*. Cambridge University Press, Cambridge, 1908.

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

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