CS6180: Lecture 12

Robert Constable

September 28, 2017

1 Lecture Summary

The first problem set asked why Brouwer changed the meaning of the standard logical operators such as "or" and "implies" and "not" and "there exists" and so forth. Everyone in the class had a good sense of what Brouwer proposed. Some students recalled his analysis of why the change matters, namely that the truth functional explanation does not account for how we actually come to know and assemble evidence for mathematical truths. Brouwer said that our understanding of mathematics is grounded in human intuitions about basic actions such as counting, organizing evidence for easy access, examining it and transforming evidence step by step.[1] In this lecture, we will look further into how we know mathematical facts. This will help explain the role of types and type theory in creating and organizing mathematical knowledge.

Computer scientists are in a good position to understand Brouwer well and extend the reach of his insights and the means of applying them to modern problems. Computer scientists need and aspire to create tools that help people experience truths and confirm conjectures or refute them. We are accustomed to learning relationships by computing. For example, to know whether a number is prime is a matter of executing a sequence of attempts to factor that number.

We are also accustomed to visual evidence as Ariel Kellison discussed in her previous lecture on Euclidean Geometry. She proved that if we have two separated points, say $a\#b$, and we are given any point c, then we can tell whether c is separated from a or from b. The point c can't be equal to both a and b since these points are separated. Therefore if we look closely around the points a and b, we will detect (with our magnifying glass) that $c\#a$ or $c\#b$. In this geometric argument, we must understand how to find and record evidence for the disjunction. We know that we can perform an inspection and collect evidence about point c. We know this because we have imagined a *computation* we can perform in principle that must yield exactly the information we need to make a decision. *Our capacity for gathering evidence by computation is at the heart of both mathematics and computer science.*

Computer scientists are comfortable *using computation to find and create evidence*, and they are accustomed to finding specific kinds of information using operations on data. In the case of

[1] In his philosophical writings [1, 3] Brouwer also considered a broader role for this approach to understanding.

geometry, some of the data is determined from a construction or from a given configuration of points by *close inspection*. Sometimes the data arises from other constructions that can be made using the points, such as constructing a circle that intersects a line or another circle in ways we can detect. Geometers draw diagrams to reveal relationships among points and lines. Visualization is a way to create relationships that reveal truths we can directly experience.

2 The role of types in creating and organizing evidence

A key reason that type theory is especially appropriate as a foundation for constructive and intuitionistic mathematics is that types provide a way to create, classify, and organize evidence. The methodology for creating types facilitates the discovery of new forms of evidence and thus for discovering new *computational truths*. The constructive type theory we created and implemented in Nuprl is known for its rich variety of types, more than in the type theories implemented by Agda or Coq.[2] Perhaps in this course we will find *a new type constructor* useful in geometry or in some other topic we investigate. It is highly likely that researchers imagine new types as we include more concepts from homotopy theory, category theory, and other branches of mathematics and computer science. Homotopy theory traces its roots to Brouwer who was also one of the creators of topology. Some of the most deep and useful types we have added over the past five years have arisen from working with the Cornell systems group on asynchronous distributed protocols. That work led us to the notion of *event structures* and *asynchronous processes* which we will discuss near the end of the course.

In Lecture 7 we discussed the critical idea of *canonical elements* of a type and canonical names for types. *It is essential to say when two canonical elements are considered equal.* Some types are abstract such as types for algebraic structures such as groups, rings, and fields. When we formalize these algebraic structures we will introduce another Nuprl type called *dependent records*. In the algebra we do not specify what the equality relation is, but we assume there is such a relation on the underlying objects that form the algebraic structure. The canonical names with a precise equality provide the basis for computation on the type. The data format must support the operations that characterize the type, such as addition for natural numbers. These operations give rise to *non-canonical expressions*, such as (12 + 5) or (17 x 15) and so forth.

In the case of Euclidean Geometry, the basic objects are *points*; however, there are no canonical objects that faithfully describe points – despite Euclid's efforts to define them, e.g. **Euclid Definition 1**: *A point is that which has no part.* This definition has been criticised for over two thousand years. It is not a paradigm for type theory. The best we might do for a canonical object is to have the canonical name pt$a$ where we take a to be an Atom or a name or something else atomic. It might be interesting to investigate this approach. What we also provide are *constructors* that build points. We can build them by bisecting segments or intersecting circles or intersecting segments and so forth.

[2] On the other hand, RedPRL and Lean are open to having rich type systems.

The rules from Lecture 9 are repeated here for easy reference with some additional comments added. We extend the theory by adding still more types, some of them have proven to be extremely useful although they are not yet implemented in other type theories. One of the most useful is the intersection type which can be used to express the new notion of *uniform evidence*.

2.1 Enumeration of types discussed

1. **universes** $\mathbb{U}$
2. **integers** $\mathbb{Z}$
3. **atoms** $Atom$
4. **empty** $Void$
5. **product** $A \times B$
6. **dependent product** $x : A \times B(x)$
7. **disjoint union** $A + B$
8. **function space** $A \to B$
9. **dependent function** $x : A \to B(x)$
10. **set types** $\{x : A | B(x)\}$
11. **quotient** $A//R$
12. **partial** $\overline{A}$
13. **intersection** $A \cap B$
14. **dependent intersection** $A : \mathbb{U}_i \cap (A \Rightarrow A)$
15. **recursive types** $rec(x.T)$

As mentioned above, in due course we will add to this list the dependent records and event structures and other types essential to fully intuitionistic type theory, which we might call *Brouwerian type theory*. This will provide at least twenty types for expressing a wide range of ideas and results important in computer science.

2.2 Comparison to set theory

It is a widely held belief that most of modern mathematics can be formalized in set theory, in particular using Zermelo/Fraenkel set theory with the Axiom of Choice [2], abbreviated as **ZFC**.

This theory is formalized using nine axioms expressed in classical first-order logic, FOL. The axioms are named as follows: Ax1 Extensionality, Ax2 Pairing, Ax3 Union, Ax4 Power Set, Ax5 Subsets, Ax6 Choice, Ax7 Infinity, Ax8 Replacement, Ax9 Foundation. FOL can be axiomatized classically with at seven axioms. So in principle, *sixteen primitive logical notions* suffice to axiomatize classical mathematics.

We can approximate the axiomatic basis for constructive type theory as having fourteen basic types with an introduction and elimination rules for each, so with approximately *twenty eight primitive logical notions*. In addition, type theory includes computation rules for using elements of the types. These are rules for reducing the non-canonical forms, about another fourteen rules for a total of about 42 rules, say 26 more rules than for ZFC. As compensation we get a programming language with precise logical rules. *The programming language helps us experience the logical truths and apply them. It enable us to experiment with alternative ways of expressing an idea.* In due course we will be able to express computational complexity concepts in type theory, and this will deepen our understanding of the evidence we use to experience mathematical truths.

Unlike with set theory, the character of type theory is to explore new types as a way to enrich the experiences we can evaluate to understand the dynamic computational reality of modern mathematics. Therefore we expect to see more types and more computation rules. We have experience with how these concepts are used to understand more deeply and control more effectively our physical reality. What we are only beginning to understand is the nature and potential of an abstract synthetic reality which we will share with our intelligent machines to enrich our notion of cyber space. We are already seeing the emergence of cyber physical systems that connect these worlds.

3 Assignment Two

Here are questions and suggestions for the second assignment.

1. Propose a *topic for your course project*. It should be related to the course and to your specific interests. It can involve using a proof assistant or writing code, but the topic should be tightly related to the ideas discussed in the lectures and readings. Provide motivation for why this topic is interesting and relevant to the course.

2. Here is a version of Markov's Principle:

$$\forall n : \mathbb{N}.(P(x) \vee \neg P(x))\&(\neg\forall x : \mathbb{N}.\neg P(x)) \Rightarrow \exists y : \mathbb{N}.P(y).$$

   (a) Explain why this is not provable in constructive type theory.

   (b) State the version of this principle expressible in iFOL and say why it is not *uniformly valid*.

3. Show how to define a number line representing an initial segment of the natural numbers in Euclidean Geometry (EG).

4. Show how to perform a geometric addition of two finite number lines.

5. Show how to divide a segment, pq into nested intervals, [[[[[...]]]]].

6. What kind of a number could this unbounded sequence of intervals represent?

References

[1] L.E.J. Brouwer. Consciousness, philosophy, and mathematics. *Proceedings of the 10th international congress on philosophy, Amsterdam*, pages 1235 – 1249, 1948.

[2] 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.

[3] A. Heyting, editor. *L. E. J. Brouwer Collected Works*, volume 1. North-Holland, Amsterdam, 1975.