CS3110 Spring 2017 Lecture 16: Discussing two topics: Synthetic Geometry and Atlantic Article

Robert Constable

	Date for	Due Date
PS5	Out on April 10	April 24
PS6	Out on April 24	May 8 (day of last lecture)

1 Synthetic Computational Geometry

We are gradually becoming familiar in the last three lectures with an approach to geometry in the style of Euclid, using abstract primitive concepts such as points, lines, polygons, congruence and so forth and with proving theorems that show the existence of geometric objects such as equilateral triangles and with performing certain constructions such as bisecting a line segment or an angle or with copying a line segment from one location to another. Euclid's proofs are intended to provide constructions, but they are not always sufficiently complete to show all of the key steps. This approach to geometry does not rely on knowing about real numbers and the analytical approach to geometry. As we started to explore this approach to geometry CS3110 in 2016, we came across an intriguing small book by Turing Award winning computer scientist Donald Knuth called *Axioms and Hulls*[8].[1] This book provides an excellent basis for a precise account of synthetic computational geometry. We have embarked on showing how this theory can be formalized in constructive type theory using the Nuprl proof assistant. Dr. Mark Bickford will illustrate the progress he has made on this in the next lecture (after spring break). In addition one of our post graduate researchers, Ariel Kellison, will show how we can formalize elements

[1]Donald Knuth is also the inventor of the Tex text editing system [7].

of Euclidean geometry in constructive type theory using Nuprl. She will explain the new geometric primitives that she has developed and explored in formalizing Euclid's *Elements* [6, 5].

1.1 Synthetic geometry primitives

Here are some of the basic concepts needed to make this approach to geometry both rigorous and constructive. We start with the abstract type of *Points*. This is not an OCaml primitive type, but it is a primitive type in our specification language, and we can easily axiomatize it in the logics of the modern proof assistants. We don't say what a point is beyond giving axioms for using them. Typically we will assign names to a finite number of points and express the properties in terms of these arbitrary names which carry no intrinsic meaning.

One implementation is the type of pairs of real numbers, e.g. the x and y coordinates in the plane. However, we can also work with an abstract type of points, $Points$. There are an unbounded number of points. The primitive relation on them is *apartness*, $a \neq b$. We say that the points are *separated*. In the previous lecture we showed how to define a notion of equality on points as well. We define $a = b$ as $\neg(a \neq b)$. This approach depends on the following axioms. We have presented these before, but now we are using slightly different notations that seem to be more expressive.

1. **Axiom**: $\forall x, y : Points.(x \neq y) \;\Rightarrow\; \forall z : Points.(z \neq x) \vee (z \neq y)$.
2. $\forall a, b : Points.a \neq b$ we write ab for a line *segment*.
3. $\forall a, b, c : Points.a \neq b \;\&\; a \neq c \;\&\; b \neq c$ we write $a - b - c$ for *separated*.
4. $\forall a, b, c : Points.a \neq b \;\&\; a \neq c \;\&\; b \neq c$ we write a_b_c for *non-strict separation*.
5. $\forall a, b, c : Points.a \neq b \;\&\; a \neq c \;\&\; b \neq c$ we write $a \neq bc$ to indicate that a is separated from the line bc.
6. $\forall a, b, c : Points.a \neq b \;\&\; a \neq c \;\&\; b \neq c$ we write a *leftof* bc for a is left-of the segment bc.
7. **Axiom** $\forall a, b, c : Points.(a \neq b \;\&\; a \neq c \;\&\; b \neq c) \;\Rightarrow\; a$ *leftof* $bc \Rightarrow a \neq bc$.

In this setting we can describe an algorithm for finding the convex hull that does not use coordinates given by numbers. We are confident that we can form the convex hull in this abstract setting, and the work of Knuth on *Axioms and Hulls* confirms this. It is not clear that Knuth believes this approach can lead to efficient convex hull algorithms, but it is a topic that we believe is worth further research. We are exploring this question and will report one result in this lecture. It is not appropriate to spend much time on new research topics in this course. It is appropriate to mention how close we are to an interesting research question. In the lecture I will mention how we came upon this research theme.

One interesting fact about this synthetic approach is that we can use it to describe the slow convex hull algorithm given in *Computational Geometry* [3]. Knuth uses it to describe another convex hull algorithm in his book. Neither of these algorithms is efficient, but they show that it is possible to solve computational geometry problems without directly using calculus. This allows us to think about these algorithms more abstractly in the style of Euclidean Geometry. It is possible that exploring this approach will lead to more intuitive specifications and to synthetic descriptions of algorithms that can be automatically compiled into efficient numerical code.

It is noteworthy that in one of Knuth's convex hull algorithms presented abstractly he uses *binary search trees.* This is a fortuitous coincidence since we will start the study of binary search trees in the very next lecture.

1.2 Slow convex hull algorithm synthetically

The method of the slow convex hull algorithm in the book *Computational Geometry* [3] can be derived using the axiomatic method of *Axioms and Hulls*, and this will open a new approach to computational geometry using proof assistants. Dr. Bickford will show how we can derive algorithms from the constructive proofs. This observation will be brought up in the second part of the lecture where we comment on the history of computer science. We will look at this idea briefly and then move on to talking about binary search trees. In the second lecture after spring break.

We see from the slow convex hull in deBerg that the specification takes as input a list of points in the region of interest. This can be presented as a numbering of points $p_1, p_2, p_3, ..., p_n$. From these points, segments are formed such as p_1p_2, p_1p_3, ... p_1p_n. The output is a list of segments that determine the convex hull. In the simple algorithm, we actually take the segments in two directions, e.g. p_1p_2, then p_2p_1 and so forth. So we will have both the upper and lower hull among these points. Then for each segment we ask whether there are points of the list defining the hull to the left. If there are none, then we know that the segment is part of the hull. If there are, then we know it is not part of the hull, and we remove it. By this simple method which can be defined with the simple left vs right primitives, we can find all the elements of the convex hull one by one. On the other hand, this is the "slow convex hull" algorithm disparaged in the book on computational geometry [3] we are studying.

Once we have listed all of these segments, we can use our primitive decidable relations to ask for each segment whether there is a point of the region defined by the list of points to the left of this segment. If there is, we know that the segment is not part of the convex hull. If there are no such points, then the segment might be part of the convex hull. We mark those points and then link them together by the relation that p_ip_j links to p_jp_k. However, this link might be tentative. We will also find a link from p_ip_k, in which case we replace the previous link by the direct one.

2 Discussing the role of logic in computer science

An article in the March 20, 2017 issue of the *Atlantic* is entitled *How Aristotle Created the Computer* has generated a lot of interest in our CS department and others around the world. It discusses the role of logic in bringing forth the computer and the discipline of computer science. On the other hand, typical of professors, my colleagues suggested ways to enrich and improve the article. I will present my ideas for enriching the story by expanding on some of the points made in the article. I can draw on personal experience here because my undergraduate advisor was Alonzo Church, one of the main players in the story, was the PhD advisor of Alan Turing and the inventor of the lambda calculus on which the programming language Lisp is based. John McCarthy, the creator of Lisp [10, 9] was a graduate student at Princeton and almost surely learned about Church's lambda calculus.[2] John McCarthy and I had a running discussion for years about the value of constructive logic versus classical logic. I recall well the time he used the argument that if we encounter alien intelligence, it will surely use classical logic not constructive logic. It was difficult to respond to this argument. The

[2]I speculate that John McCarthy did not take Church's course on type theory, and that is why Lisp does not have a type system.

way these comments are related to the article is to mention a sentence of the article and then inject after it comments related to the course or computer science. Thus various sentences from the Atlantic article are *quoted below in italics* to structure a discussion of it. The url provided by the *Atlantic* for the full article is available on the course web site.

"THE HISTORY OF computers is often told as a history of objects, from the abacus to the Babbage engine up through the code-breaking machines of World War II. In fact, it is better understood as a history of ideas, mainly ideas that emerged from mathematical logic ..."

Logic has roots in the writings of Aristotle which were considered by the Greeks and Romans as guides to debating and rhetoric, an important skill in the life of the leaders of Greek and Roman society. Logic was a practical discipline in that sense. The ultimate basis for logical truth could be traced to Plato's notion of an ideal world and the truth about ideal objects like numbers that we could not directly see and experience. That was a less practical feature of Aristotle's logic.

"The evolution of computer science from mathematical logic culminated in the 1930s, with two landmark papers: Claude Shannons A Symbolic Analysis of Switching and Relay Circuits, and Alan Turings On Computable Numbers, With an Application to the Entscheidungsproblem. In the history of computer science, Shannon and Turing are towering figures, but the importance of the philosophers and logicians who preceded them is frequently overlooked."

The articles mentioned are these.

1. Church: A set of postulates for the foundation of logic, [1].

2. Turing: Computability and lambda-definability, [13].

Not mentioned is that Turing was working on a famous open problem posed by Hilbert that Church had recently solved, so his British academic advisors suggested that Turing visit Princeton and learn of Church's results. Turing wrote his PhD thesis under Church's supervision. For his thesis he "built a translator from the Church's lambda calculus to his machine model", which we now know as *Turing Machines.* This in a way was the first compiler. Turing then turned his attention to writing about type theory, but this work was interrupted by the war. We did not know of Turing's top secret work during the war until at least the 70's. I remember how amazed we were to learn this story which is now captured in a movie as well as biographies [12].

"Aristotle also defined a set of basic axioms from which he derived the rest of his logical system: An object is what it is (Law of Identity) No statement can be both true and false (Law of Non-contradiction) Every statement is either true or false (Law of the Excluded Middle)"

These axioms were not intended to describe how people actually think (that would be the realm of psychology), but how an idealized, perfectly rational person ought to think. Yet, the most modern proof assistants such as Nuprl and Coq do not think this way. Nuprl is based on Brouwers insight which is more general and from which Aristotles can be explained.

We have mentioned already in this course that by 1908 L.E.J. Brouwer challenged the third of the claims listed above. Many computer scientists have accepted this challenge as highly relevant. Brouwer said that we don't know what it means to say that every statement is either true or false. He claimed that we come to know mathematical truths by experiencing them not by probing the *Platonic Reality* that the Greeks imagined. Intuitionists prefer to stress that by experience attempting to compute using various concepts we come to understand them and come to know what constitutes *evidence* for a mathematical claim. We understand various categories or *types of evidence*. This becomes the *evidentiary basis of mathematical truth*, and it is naturally captured in type theory as we are experiencing for ourselves using OCaml to provide the evidence and the experience of using it.

Although ostensibly about geometry, Euclid's Elements *became a standard textbook for teaching rigorous deductive reasoning. (Abraham Lincoln once said that he learned sound legal argumentation from studying Euclid.) In Euclids system, geometric ideas were represented as spatial diagrams. Geometry continued to be practiced this way until Ren Descartes, in the 1630s, showed that geometry could instead be represented as formulas. His Discourse on Method was the first mathematics text in the West to popularize what is now standard algebraic notation, x, y, z for variables, a, b, c for known quantities, and so on.*

We have seen that there is significant value in the way Euclid approached geometry. His account is axiomatic and abstract. We also called it *synthetic geometry* in contrast to the analytic geometry that arises when we use the *Cartesian* coordinate system, named after Descartes. In our course we are exploring the contrast between analytic geometry based on constructive analysis and synthetic geometry based on *computationally meaningful axioms about points and lines.*

Turing worked in a tradition stretching back to Gottfried Leibniz, the philosophical giant who developed calculus independently of Newton. Among Leibnizs many contributions to modern thought, one of the most intriguing was the idea of a new language he called the “universal characteristic.” It could represent all possible mathematical and scientific knowledge. Reading Leibniz’s ideas is inspiring even now. His vision has inspired generations of philosophers, mathematicians and now computer scientists. A critical step toward this vision has been the development of modern type theory and its implementation in proof assistants such as Coq, Nuprl, Agda and others in the pipeline. Here is what we wrote in the 1984 article *Writing programs that construct proofs* [2]:“In the time of the Greeks, geometers were already building machines to help them with derivations. Continuous and sustained interest in providing mechanical aids to reasoning can be traced to the seventeenth century. Gottfried Leibniz is popularly believed to have contributed to symbolic logic in striving to mechanize reasoning, and while his technical contributions in this subject were minor his vision and the authority which his stature accorded it are with us today. His words still kindle and interest little diminished by the naivete of their details. Here is what he wrote:

“A term is a subject or predicate of a categorical proposition .... Let there be assigned to any term its symbolic number, to be used in calculation as the term itself is used in reasoning. I chose numbers whilst writing in due course I will adapt other signs .... For the moment, however, numbers are of the greatest use ... because everything is certain and determinate in the case of concepts, as it is in the case of numbers. The one rule for discovering suitable symbolic numbers is this; that when the concept of a given term is composed directly of the concept of two or more other terms, then the symbolic number of the given term should be produced by multiplying together the symbolic numbers of the terms which compose the concept of the given term. In this way we shall be able to discover and prove by our calculus at any rate all the propositions which can be proved without the analysis of what has temporarily been assumed to be prime by means of numbers. We can judge immediately whether propositions presented to us as proved, and that by means of numbers. We can judge immediately whether propositions presented to us are proved, and that which others could hardly do with the greatest mental labor and good fortune, we can produce with the guidance of symbols alone .... As a result of this, we shall be able to show within a century what many thousands of years would hardly have granted to mortals otherwise.”

As Bertrand Russell famously quipped: *Mathematics may be defined as the subject in which we never know what we are talking about, nor whether what* *we are saying is true.*

Russell's formal approach directly contrasts with Brouwer's intuitionistic approach in which mathematics is about our ways of discovering and using insights about number and computation. Computer science has perfected the treatment of formal syntax as we see in defining and recognizing the syntax of programming languages such as OCaml. Computer science has also made precise Brouwer's insight that we experience mathematical truths in many cases by mental computation confirmed by its implementation in a high level programming language.

"An unexpected consequence of Freges work was the discovery of weaknesses in the foundations of mathematics. For example, Euclids Elements *considered the gold standard of logical rigor for thousands of years –turned out to be full of logical mistakes. Because Euclid used ordinary words like line and point, he –and centuries of readers – deceived themselves into making assumptions about sentences that contained those words."*

We have seen in this course how to make some of Euclid's concepts logically precise and how to give concrete meaning to his geometric constructions, e.g. creating an equilateral triangle and bisecting angles and so forth. So we're still working on Euclid *23 hundred years later.*

"To give one relatively simple example, in ordinary usage, the word line implies that if you are given three distinct points on a line, one point must be between the other two. But when you define line using formal logic, it turns out between-ness also needs to be defined – something Euclid overlooked. Formal logic makes gaps like this easy to spot."

This is an example which we explicitly examined in the course to show the need to make Euclid more precise if we are to use his examples in modern constructive mathematics.

Turing was working in a tradition stretching back to Gottfried Leibniz, the philosophical giant who developed calculus independently of Newton. Among Leibniz's many contributions to modern thought, one of the most intriguing was the idea of a new language he called the universal characteristic that, he imagined, could represent all possible mathematical and scientific knowledge. Inspired in part by the 13th-century religious philosopher Ramon Llull, Leibniz postulated that the language would be ideographic like Egyptian hieroglyphics, except characters would correspond to atomic concepts of math and science. He argued this language would give humankind an instrument that could enhance human reason to a far greater extent than optical instru-

ments like the microscope and telescope. He also imagined a machine that could process the language, which he called the calculus ratiocinator

As Bertrand Russell famously quipped: Mathematics may be defined as the subject in which we never know what we are talking about, nor whether what we are saying is true.

In stark contrast, the topologist Brouwer was saying that all mathematical truth is experienced truth based on intuition and mental constructions. This conception of mathematics actually leads to a new logic that is not mentioned in the article. It is called either *constructive* logic or *intuitionistic logic*, and it is playing an increasing role in programming languages. The polymorphic logic of OCaml that we have studied in this version of the course is an intuitionistic propositional logic. The *Atlantic* article ends with this increasingly familiar theme.

This would be a fitting second act to the story of computers. Logic began as a way to understand the laws of thought. It then helped create machines that could reason according to the rules of deductive logic. Today, deductive and inductive logic are being combined to create machines that both reason and learn. What began, in Booles words, with an investigation concerning the nature and constitution of the human mind, could result in the creation of new mindsartificial mindsthat might someday match or even exceed our own.

This has been a major goal of the field of *Artificial Intelligence* (AI) since its beginning, to create machines that can reason, learn, and participate in discovery. The book series *Machine Intelligence* [11] included many articles on the task of creating machines that can help us do mathematics and assist in the creation of proofs. This fascinating quote conveys something about the stakes in this area of computer science:

"In the game of life and evolution, there are three players at the table: human beings, nature, and machines. I am firmly on the side of nature. But nature, I suspect, is on the side of machines" wrote George B. Dyson [4]. This kind of historical account in the Atlantic reminds us of one of the great disadvantages of being dead, one can no longer contribute. Once AI has seen a way to build intelligent machines, it might turn attention to the notion that certain aspects of a particular human's behavior might live on. Characteristics of thought that can be embodied in tactics, strategies and plans used by an intelligent machine will provide a virtual presence that continues to learn and improve. We can imagine a machine that learns many ways to attack a problem by working with a large group of human users or by working with one outstanding problem solver and capturing his or her modes of thought.

References

[1] Alonzo Church. A set of postulates for the foundation of logic. *Annals of mathematics, second series*, 33:346–366, 1932.

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

[3] M deBerg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry*. Springer, 2008.

[4] George B. Dyson. *Darwin Among the Machines: The Evolution of Global Intelligence*. Perseus Books, 1997.

[5] Euclid. *The Elements*. Green Lion Press, Santa Fe, New Mexico, 2007.

[6] Euclid. *Elements*. Dover, approx 300 BCE. Translated by Sir Thomas L. Heath.

[7] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, MA, 1984.

[8] Donald E. Knuth. *Literate Programming*. C.S.L.I Publications, 1992.

[9] 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.

[10] J. McCarthy et al. *Lisp 1.5 Users Manual*. MIT Press, Cambridge, MA, 1962.

[11] Donald Michie. *Machine Intelligence 3*. American Elsivier, New York, 1968.

[12] Peter Millican and Andy Clark. *The Legacy of Alan Turing, Vol. 1: Machines and Thought*. Oxford University Press, New York, 1996.

[13] A. M. Turing. Computability and λ-definability. *Journal of Symbolic*, 2:153–63, 1937.