Math 29: Introduction and History of Computability

March 28th, 2022

1 Course Structure

Welcome to Math 29! The course syllabus and relevant information not included here can be found on the course website.

Your grade will be made up of class participation (15%), weekly homeworks (30%), a midterm (25%), and a final (30%). Showing up to class, answering questions, asking questions, and participating in any group activities/discussions all count towards class participation. Anyone who attends and engages in class regularly does not need to worry about the in-class grade. Any students who are potentially falling behind will be gently warned before they lose any points.

Homeworks will be assigned each Wednesday and due by midnight the following Wednesday. They will cover material from the assigned week of lectures. For example, Homework 1 will be assigned on Wednesday March 30th and will cover the lecture material from Monday the 28th, Wednesday the 30th, and Friday April 1st. Returned homework can be revised and re-submitted for partial credit, with the second and subsequent resubmissions decreasing the possible score. Section 2.6 in the textbook contains some helpful advice for writing proof.

The midterm and final exams will follow the same format. In-class, there will be a knowledge quiz composed true or false, multiple choice, and short answer questions. Each of these will account for 10% of your final grade. Both exams will also have a take-home component, which will be worth 15% and 20% respectively. The take-home midterm will take the place of the homework that would otherwise be due on May 4th. The take-home final will be due at the end of the final exam period. **Unlike homework assignments, take-home exams cannot be resubmitted.**

On Thursdays, during the X-hour, Ben Logsdon will be running extra sessions covering course material. These may include a mixture of lectures, worksheets, problem solving sessions, and more. Attendance and participation will count towards your class participation grade.

2 History

For over a thousand years, we've known how to solve quadratic equations. In particular, the quadratic equation, explicitly gives the roots of the equation $ax^2 + bx + c = 0$ as

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

This formula not only enables us to calculate what the solutions are, but enables us to easily determine whether or not the solutions are real or complex by checking the sign of the discriminant. Of course, curious mathematicians then wondered: is there a general formula for solving other polynomials? Over the years, similar, albeit significantly more complicated, formulas were found for third and fourth degree polynomials.

The turn of the nineteenth century saw the Abel-Ruffini theorem, which states that no such general equation exists for any higher order polynomial. In other words, for fifth degree and higher, we don't have a clean, uniform method for finding roots of polynomials. This is an early example of a "non-computability" result.

Questions of what we could and could not find an algorithm or "process" for continued to be of interest to mathematicians. Ada Lovelace and Charles Babbage created and implemented what would later come to be considered the world's first computer in the mid 19th century. In 1900, David Hilbert gave a list of 23 famous open problems. Of these, two would turn out to have strong connections to the this idea of having algorithms that can determine information: the second and the tenth.

Hilbert's second problem was to prove that arithmetic, the most basic part of mathematics, is consistent. In other words, show that it is not possible to prove that $0 = 1$. Clearly, if this were to happen, our entire understanding of numbers would collapse: not only is $0 = 1$, but then $0 = 1 + 0 = 1 + 1 = 2$. Continuing in this fashion, we see that $0 = 1$ would result in EVERY number being equal to 0. Of course, this seems impossible because we know from experience that $0 \neq 1$. The danger here is the **principle of explosion**: if we can prove something false, then we can prove everything. If we can prove something false, then the system collapses. However, the seminal work of Kurt Gödel in the 1930's showed that it is impossible to prove that we can avoid proving $0 = 1$. Expansions of this result later proved something even stronger: if we are given an algorithm that lists out some basic properties of arithmetic, there is no algorithm that can then list out a complete list which is provable from those properties.

Hilbert's tenth problem concerns diophantine equations: polynomials in finitely many variables with integer coefficients. Hilbert hopes to find a general

process, or algorithm, by which we can determine whether or not it has an integer root. We know from the Abel-Ruffini theorem that even in the case of one variable, we cannot always hope to find the roots. However, can we even determine if they exist? The answer is still no, as proven in latter half of the 20th century by a collective of mathematicians including Julia Robinson and Yuri Matiyasevich.

Much of this work would not be possible without a formal treatment of the mere idea of an algorithm, or a process, by which information is decided upon. Multiple different yet equivalent formal definitions of what it meant to be "recursive," later "computable," were given and studied by the likes of Alonzo Church, Stephen Kleene, and Alan Turing. These gave a mathematical statement about what algorithms are, what they can do, and more importantly, what they cannot do. This gave rise to the Church-Turing thesis (also known by many other variations): that this formal notion of computability is exactly the correct one to match our intuition. In other words, any piece of information that we might one day devise a machine or algorithm to discern would in fact be computable in the formal sense above. (Current models of quantum computation are no exception: they provide outstanding benefits to efficiency and speed, but do not fundamentally alter that which could be computed traditionally with unlimited resources.)

Ironically, much of computability theory is the study of things which are **not** computable. We will talk about many such sets in this course, the most famous of which is the **halting set**. Basically, no computer can read arbitrary computer code and figure out what it will do! (One reason why antivirus software can never be perfected.) Emil Post and others spent much of the mid 20-th century describing a variety of other sets with different properties which cannot be described by an algorithm, i.e. sets that are not "computable."

Given some piece of information that is not computable, the next question becomes: what sort of secondary information can one use in order to determine it? This leads to **relative computability** or **oracle computability**: a hierarchy which breaks down information based on how strong is is from a standpoint of computational power. More knowledgeable sets, which can compute many pieces of noncomputable information, are high on the list, while less knowledgeable sets which can only compute a few pieces of noncomputable information are much lower.