

What Is Computability Theory?

An Introduction to Math 29

Foundation

We have an intuitive notion of what it means for something to be computable; there is an algorithm or recipe or mechanical means to carry it out.

We need an abstract definition if we are to have any hope of proving results about computability in general.

Some options:

- ▶ Declare *computable* to mean *executable by a program written in Java* (or another fixed language).
- ▶ Declare *computable* to include certain functions and be preserved by composition and similar operations.

Foundation II

We'll explore definitions following both of those models (*Turing machines* and *partial recursive functions*).

They look very different but define the same class of functions.

We have the **Church-Turing Thesis**: these definitions actually capture the intuitive notion of computability.

Not a provable statement!

Early Steps

We'll need some tools to work with computability.

- ▶ *Coding*: a way to work with objects that are not natural numbers.
- ▶ *Enumeration*: a way to assign indices to Turing machines.
- ▶ The *universal Turing machine*: a single program that does all possible computations.
- ▶ *Parametrization* and the *Recursion Theorem*: results about indices of machines.

The Halting Problem

The first *noncomputable* object we'll see is the Halting Problem, the set of indices i such that the i^{th} Turing machine halts when given input i .

This is a key example: many noncomputability proofs (including the proofs of unsolvability we'll see) work by showing computability of the object at hand would imply computability of the Halting Problem.

A Hierarchy

The use of the word “object” in previous slides is because we can think of functions, sets, and other mathematical objects as being computable or noncomputable. (We will start with functions and generalize to sets.)

Suppose our Turing machine has a CD that lists the contents of a set A . We call it an *oracle* machine – if A is noncomputable, then the machine/oracle combo might not be replicable by a machine without an oracle.

This allows us to define an ordering on sets: $A \leq_T B$ (T for Turing) if A can be computed by a machine with oracle B .

The Halting Problem Again

As it turns out the Halting Problem is the top of a natural collection of sets, the *computably enumerable* (c.e.) sets.

A set is c.e. if its elements can be listed out, not necessarily in order. Call the Halting Problem H ; it is c.e. and for all c.e. A , $A \leq_T H$.

We can also show the hierarchy goes up forever; even if we give all our Turing machines H as an oracle there are things they still can't compute, and so on.

Ending Points

From there the course content will depend on remaining time and class interests. We can look at some current areas of research, including

- ▶ the structure of sets under \leq_T
- ▶ reverse mathematics: the classification of theorems according to the strength of the axioms required to prove them
- ▶ algorithmic randomness: like computability, randomness is something we have an intuitive notion of. We can use computability to define it mathematically and investigate the consequences.