UCT

Complexity Theory

Klaus Sutner

Carnegie Mellon University

Spring 2024

Dramatis Personae

- Prof:
  Klaus Sutner sutner@cs.cmu.edu

- TAs:
  Russel Emerine remerine@andrew.cmu.edu
  Nicholas Kocurek nkocurek@andrew.cmu.edu

- Course secretary:
  Rosie Battenfelder rosemary@cs.cmu.edu

Web and Communication

Course Page http://www.cs.cmu.edu/~uct

Edstem https://edstem.org

Make sure to read the syllabus posted on the course site. We will assume that you are familiar with all the rules set out in the document. If you have questions, ask (Ed seems like a good venue, you're probably not the only one).

Graduating Seniors

There are 2 in this class. I assume you need this course to graduate.

Make sure to stay on top of things.

If there is an issue, talk to me early on—not when it's too late to fix anything.

Learning Style

Far and away the most important challenge is for you to develop your **intuition**.

Technical details are necessary and indispensable, but intuition comes first—by a long shot. A very, very long shot.

This is all old hat, but in computability/complexity theory it is even more important than in other areas. The reason: the technical details are often quite messy, it's critical to have a clear intuitive sense of where things are going.

Definitions

intuition understand what the concept means, what it's purpose is

formalization pin things down in a semi-formal way

examples some objects that the definition applies to

counterexpl some objects where it does not, but almost

results the basic theorems associated with the concept

Theorems

intuition understand what the result says, its meaning

formalization pin things down in a semi-formal way

examples situations where the theorem applies

counterexpl situations where the theorem does not apply, but almost

connections other related results and methods

Proofs

intuition understand the objective precisely, develop a battleplan

formalization refine the argument to a semi-formal level

optimality what happens if we change hypotheses/conclusions

method what is the big idea in the proof

explanation why is the result true

Comments

The hedge **semi-formal** is just a reminder that we are not concerned with truly formal arguments, using prove checkers and theorem provers. This is out of the question, it leads straight to insanity. Like everyone else, we will be dealing with **proof sketches**.

The *explanation* item is particularly important: ideally, a good proof provides an explanation why the result is true, why it makes sense and how it fits into a larger framework.

Unfortunately, added explanatory value seems to clash hopelessly with formality. Some computer supported proofs are infuriatingly meaningless. Another reason to stick to proof sketches.

Why the Fuss?

A lot of the arguments in complexity theory are based on reasoning about **Turing machines**. On the face of it, these devices are fairly simple, just finite state machines with simple memory attached—a glorified type writer, as it were.

True, but the devil is in the detail. Just about all arguments about TMs look like

> . . . can construct a Turing machine that does the following . . .

No one ever carries out the construction with any level of precision, it's all an appeal to intuition. And it can go wrong.

Books

. . . are hopelessly outdated and obsolescent (only partially kidding). You are often better off doing a little web search for a particular topic.

We have no textbook. If you still feel very attached to books, below are some plausible candidates.

There is also a lot of good material on the web (stick to reputable sources).

- **M. Sipser**, *Introduction to the Theory of Computation*
- **O. Goldreich**, *P, NP and NP-Completeness*
- M. Garey, D. Johnson, *Computers and Intractability*
- B. Barak, S. Arora, *Computational Complexity: A Modern Approach*
- C. Papadimitriou, *Computational Complexity*
- A. L. Selman, S. Homer, *Computability and Complexity Theory*
- L. A. Hemaspaandra, M. Ogihara, *The Complexity Theory Companion*
- **C. Moore, S. Mertens**, *The Nature of Computation*

Computability

- **H. Enderton**, *Computability Theory: Introduction to Recursion Theory*
- R. Soare, *Recursively Enumerable Sets and Degrees*
- H. Rogers, *Theory of Recursive Functions and Effective Computability*
- P. Odifreddi, *Classical Recursion Theory, Volumes I/II*
- S. Cooper, *Computability Theory*

Aka **classical recursion theory** CRT or RT.

Take a close look if you want to go to grad school.

Computability First

We will start with a brief recap of general, old-fashioned computability.

Then we switch to complexity in the modern sense. As you will see, there is a lot of repetition and analogy—though things invariably get messier in the complexity world. So enjoy the clean, simple world of computability while it lasts.

What you should know already:

- At least one model of computation (Turing machines).
- Coding functions (computable bijections $\mathbb{N}^{<\omega} \to \mathbb{N}$).
- Existence of universal machines.
- Existence of semidecidable but undecidable sets (Halting).
- Reductions between Halting and other problems.

The Big Questions

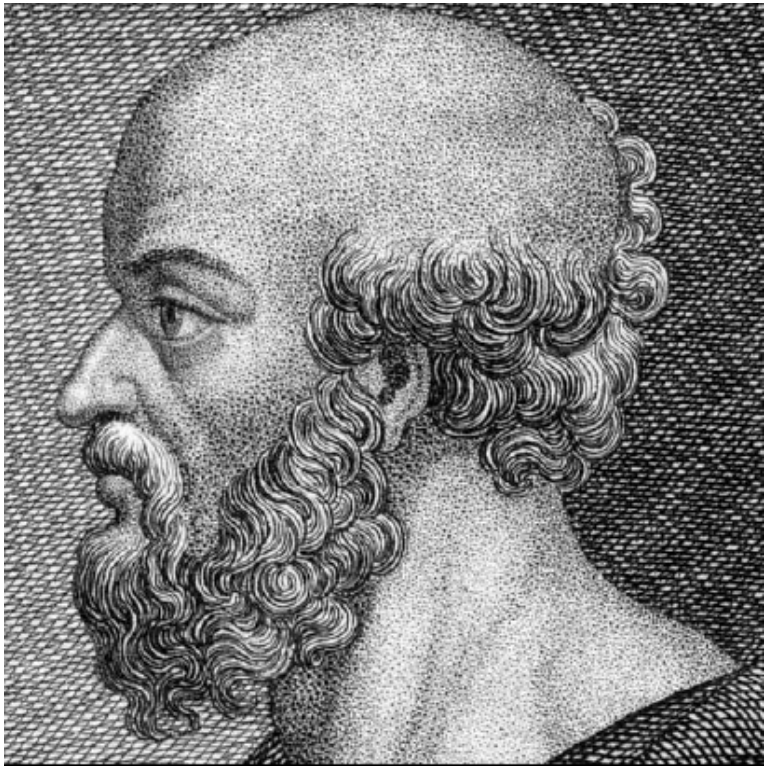
- What is computability?
- What is feasible computation?

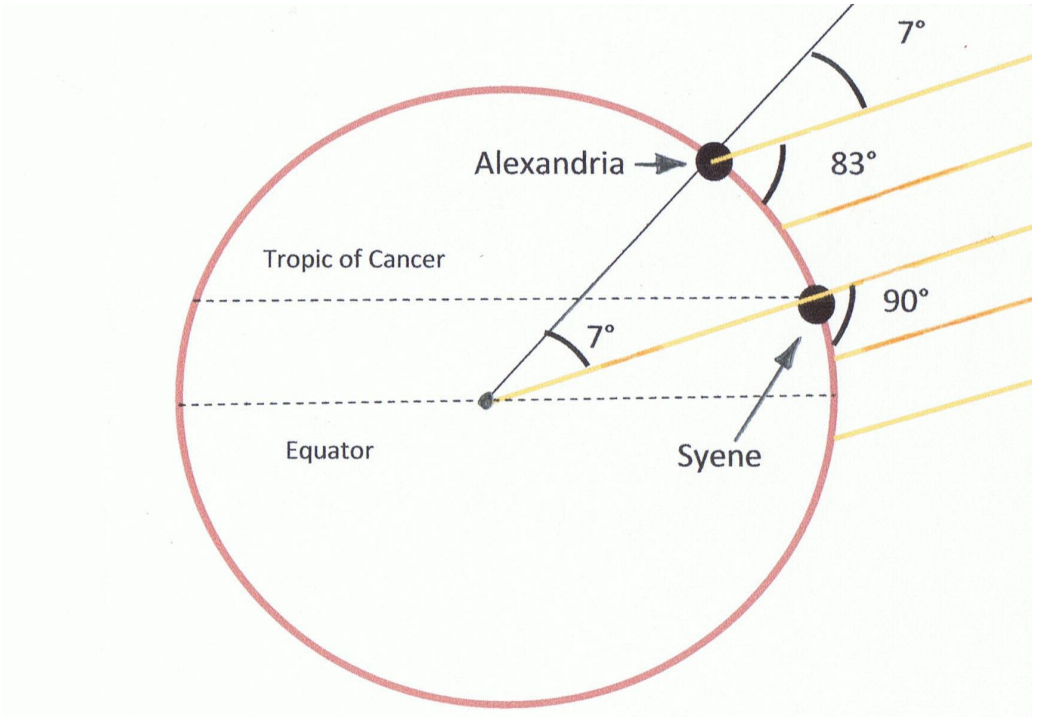
For most of its history, mathematics was focused on computation: solving equations, measuring areas and volumes, constructing geometric objects, testing primality, and so on. A more recent example: high precision numerical integration. This stuff, while technically complicated, is often very intuitive and tangible.

Ancient Pre-History

Early mathematics was very much focused on computation (Plimpton 322, about 1800 BCE). See Pythagorean triples for an explanation.

Eratosthenes





Calculated distance from earth to sun around 240 BCE, error may have been as low as 1%. Also calculated the diameter of the sun, not as accurate.

Around 1530, N. F. Tartaglio managed to solve equations of the form $x^3 + ax^2 + b = 0$.

Here is one of the solutions:

$$\frac{1}{3}\left(\sqrt[3]{-a^3 + \frac{3}{2}(-9b + \sqrt{3}\sqrt{b(4a^3 + 27b)})} + \frac{a^2}{\sqrt[3]{-a^3 + \frac{3}{2}(-9b + \sqrt{3}\sqrt{b(4a^3 + 27b)})}} - a\right)$$

Even just verifying that this is a solution is not so easy, never mind finding it in the first place.

A Misconception

There is often an unspoken assumption that the theory of computation is concerned with this type of calculation, just in a more rigorous and organized fashion, and with a view towards digital computers.

> Nothing could be further from the truth.

The theory of computation was developed to overcome problems in the foundations of mathematics. It is a central part of mathematical logic. There are some connections to applied computation, but that is almost an accident.

Grundlagenkrise

Why on earth would there be any problems in the foundations of mathematics?

After all, math is perfect, precise, rigorous, timeless, eternal ...

Not so fast. In the good-old-days one might have made such claims with a straight face, but in the late 19th, early 20th century things start to go sideways.

Classical Period

Breakthrough

The invention of calculus in the late 17th century was a huge conceptual and practical breakthrough. The modern world is simply unthinkable without calculus. Needless to say, it works perfectly well.

And yet, there were also the beginnings of trouble: calculus was based on nebulous infinitesimals, vanishingly small quantities that somehow also were real numbers (though, at the time, no one could define the reals either)[†]. Calculus is so natural and intuitive that seasoned practitioners can get great results—even without weight-bearing foundations.

Leibniz clearly realized there was a problem, but had nowhere near the right tools to fix the issues, that would take till the 1960s with the discovery of non-standard analysis by A. Robinson.

[†] . . . the Ghosts of departed Quantities. *The Analyst*, Bishop Berkeley, 1734.

Euler's Intuition

Leonhard Euler (1707–1773) had essentially perfect intuition and could concoct arguments that were eminently plausible, and led to correct results, but were exceedingly difficult to justify in the modern sense. Here is an example:

Problem: Find a way to calculate e^x for real x.

Wlog $x > 0$. For x reasonably small we can write

$$e^x = 1 + x + error$$

with the error term being small. Alas, we have no idea what exactly the error is.

Infinitesimals

Euler considers an infinitesimal $\delta > 0$. This simplifies matters greatly:

$$e^{\delta} = 1 + \delta$$

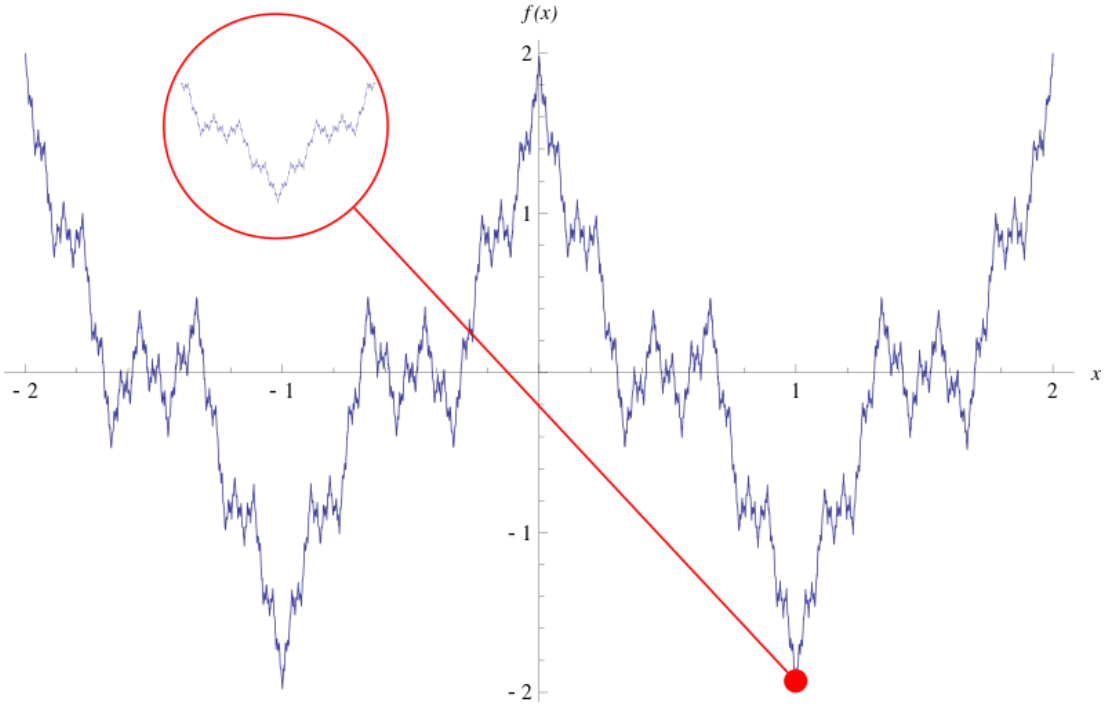
This identity is justified by Leibniz's lex homogeneorum transcendentalis, the transcendental law of homogeneity: proof of the law is by higher authority. Again, this is not Leibniz's fault, he was way ahead of his time.

Once we accept the last identity, we can calculate happily:

$$
\begin{aligned}
e^x &= (e^\delta)^{x/\delta} \\
&= (1+\delta)^{x/\delta} \\
&= 1 + \binom{x/\delta}{1}\delta + \binom{x/\delta}{2}\delta^2 + \dots \\
&= 1 + x + 1/2\, x(x-\delta) + \dots \\
&= 1 + x + 1/2\, x^2 + \dots \\
&= \sum_{i \geq 0} x^i/i!
\end{aligned}
$$

19th Century

Things got worse during the 19th century with the discovery of rather bizarre objects like Weierstrass's continuous function that is nowhere differentiable.



Even worse, the function is just a Fourier series, a device beloved by engineers. Trigonometry creates monsters.

Towards Rigor

Once it became clear that one needed to be very, very careful to avoid "results" that were plain wrong, a number of people started to work on the development of solid foundations for analysis.

- Cauchy and Dedekind gave precise definitions of the reals (Cauchy sequences, Dedekind cuts).
- Weierstrass used limits and ε/δ proofs.
- Cantor developed a fine-grained theory of Fourier analysis.

A combination of logic (G. Frege, G. Peano) and set theory (G. Cantor) seemed like the right mechanisms to build a completely rigorous foundation of math. One would just have to reconstruct math in these frameworks, and everything would be fine.

Disaster Strikes

Alas, that was not meant to be, These tools are excellent, but used naively they lead to a whole number of other problems, even contradictions.

A typical paradox is produced by Russell's self-contradictory set

$$S = \{ x \mid x \notin x \}$$

This set S is rather bizarre and seems utterly useless, but that's beside the point: in Frege's system this set was perfectly legitimate, no less so than, say, the set of prime numbers.

A Silver Lining

What was terrible for Frege was actually good for math and CS, though. Russell's observation directly lead to two fundamental developments:

- Axiomatic Set Theory
  Zermelo with a critical contribution by Fraenkel produced the standard reference system for math: Zermelo-Fraenkel set theory, nowadays usually augmented by the Axiom of Choice. Chances are, all the math you ever learned was rooted in **ZFC**.

- Type Theory
  Russell himself developed type theory as a way to rule out problems like his paradoxical set. His system failed as a foundation of math, but types have become critical in TCS.

An Earthquake?

A naive person might think that Russell's discovery would have rocked a lot of boats; actually, it might have capsized them.

But, the establishment has a standard way to deal with crises: ignore them. Just about everybody kept on putzing around happily, as if nothing had ever happened.

A few people like David Hilbert took the problem seriously and decided to do something about it. On the other hand, some, like Henri Poincaré, actively resisted the idea of trying to fix a system they didn't think was broken. Notably, these two were the top mathematicians around 1900.

Poincaré and Hilbert

> Formerly, when one invented a new function, it was to further some practical purpose; today one invents them in order to make incorrect the reasoning of our fathers, and nothing more will ever be accomplished by these inventions.

- Axiomatization
  Updates Euclid, introduces the de facto modern standard.
- Hilbert's Program
  Construct a foundation of math by strictly finitary means.
- Entscheidungsproblem
  Find a mechanical, definite, finite procedure to determine the validity of any formal statement in an axiomatic system.

The last item directly translates into CS: the challenge is to find a decision algorithm, essentially for all of math.

The other two are also closely connected, but that is not as obvious. Note the constraint *strictly finitary*, this is the place where computation enters the picture.

Axiomatization has become so utterly and completely standard that it is almost no longer noticeable: any halfway serious development will have a clear axiomatic foundation, and proving anything means to derive it from the axioms (though emphatically not in a strictly formal manner).

There may be discussions about how to best axiomatize a particular domain, or even whether certain axioms are justified (axiom of choice in set theory), but there is no serious discussion about whether axiomatization is desirable and even necessary.

2: Hilbert's Program

In the 1920s, in response to paradoxes and intuitionistic lunacy, David Hilbert proposed a program to salvage all of mathematics. In a nutshell:

> Formalize mathematics and concoct a finite set of axioms that is is free from contradictions (consistency) and strong enough to prove all theorems of mathematics (completeness). Establish these properties by strictly finitary means.

Consistency means that the system is not self-contradictory, one cannot prove both an assertion and its negation.

Completeness means that all true statements can be proven, so the axioms are strong enough.

So, in a system that is both consistent and complete, we can derive exactly all true statements, the ideal scenario.

Proofing Things

If we want to make precise what we mean by a purely mechanical procedure that yields a result after finitely many steps we wind up with the notion of computation.

In particular, to verify that a given string really represents a valid proof, we want a proof checker, a decision algorithm that takes strings as inputs and returns yes or no depending on whether the given string really is a valid proof in some precisely defined formal system.

This has an important consequence: once we know how to check proofs, we automatically get a proof enumeration algorithm: just run through all possible strings (say, in length-lex order), and filter out the ones that constitute valid proofs.

Checkers versus Provers

For the record: to build a theorem prover is a much harder problem: we want an algorithm that takes as input a formula, and determines whether it has a proof in the system.

This works for some systems, and fails catastrophically for others, to the major dismay of Hilbert.

As we will see, the gap between checkers and provers is very, very similar to the $\mathbb{P}$ versus $\mathbb{NP}$ problem. In fact, Gödel had the basic concept already in the 1950s.

Promising Start

Initially good progress was made towards identifying logical systems that could provide the necessary deductive machinery, without worrying too much at this point about axiomatizing interesting areas such as arithmetic.

- completeness of propositional logic (Boolean logic, Ackermann 1928),
- completeness of predicate logic (aka first-order logic, Gödel 1930).

Propositional logic is far too weak to support interesting mathematics (it will play a major role in complexity, though), but first-order logic seemed very well suited for Hilbert's project. For example, ZFC is typically described as a first-order theory.

A Bombshell

Alas, in 1931, Kurt Gödel essentially wrecked Hilbert's program:

> Any Hilbert system that can express basic arithmetic is always incomplete or inconsistent.

So any consistent Hilbert system is always incomplete: there are some true statements that simply cannot be proven in the system.

Worse, the problem cannot be fixed by simply adding the unprovable statement: another true but unprovable statement will pop up in the new system. The problem is not that the designer of the system was careless.

> The Entscheidungsproblem is solved when one knows a procedure by which one can decide in a finite number of operations whether a given logical expression is generally valid or is satisfiable. The solution of the Entscheidungsproblem is of fundamental importance for the theory of all fields, the theorems of which are at all capable of logical development from finitely many axioms.
>
> D. Hilbert, W. Ackermann
> Grundzüge der theoretischen Logik, 1928

In modern terminology: find a decision algorithm for statements of mathematics in any axiomatic formal system.

Note that no proof was required, just a one-bit answer.

The Nub of the Problem

Hilbert's Entscheidungsproblem comes down to the following.

> Given a sentence of first-order logic, determine whether the sentence is valid.

Validity here means true in all possible interpretations (all possible structures over which the sentence makes sense). By the completeness theorem, that is equivalent to provability in some suitable formal system.

But provability is "merely" a syntactical notion, it might well be the case that one can decide whether a proof exists or not (truth over all possible structures seems a lot more complicated).

Another Bombshell

Alas, that did not work out, either[†].

Theorem (Turing 1936)

The Halting problem for Turing machines is undecidable.
As a consequence, first-order logic is also undecidable.

In fact, a fairly small fragment of arithmetic known as Robinson arithmetic already suffices (just successor, addition and multiplication, no induction).

A finite set of fairly simple arithmetic axioms is enough to implement Turing machines.

[†]Church had another proof based on his λ-calculus at the same time.

Gödel vs Turing

Gödel has shown, in essence, that in any reasonable formalization of arithmetic there a assertions that can neither be proven nor refuted[†].

If the opposite were true, every assertion is either provable or refutable, then the Entscheidungsproblem would be solvable: just enumerate proofs until you either get to the assertion itself, or its negation.

Turing (and independently Church) showed that the Entscheidungsproblem is indeed unsolvable, even when restricted to a fairly weak subsystem of math.

[†]In the olden days this was expressed by saying "the assertion is undecidable in arithmetic." Hence the title of Gödel's seminal paper: *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme,* We will avoid this terminology like the plague.

> The precise and unquestionably adequate definition of the general concept of formal system [made possible by Turing's work allows the incompleteness theorems to be] proved rigorously for every consistent formal system containing a certain amount of finitary number theory.
>
> K. Gödel, 1963

Models of Computation

- K. Gödel: primitive recursive
- A. Church: λ-calculus
- J. Herbrand, K. Gödel: general recursiveness
- **A. Turing: Turing machines**
- S. C. Kleene: μ-recursive functions
- E. Post: production systems
- H. Wang: Wang machines
- A. A. Markov: Markov algorithms
- M. Minsky; J. C. Shepherdson, H. E. Sturgis: register machines

The models are listed roughly in historical order.

Except for primitive recursive functions[†], these models are all equivalent in a strict technical sense.

This does **not** mean that they are equally intuitive or compelling. For example, unless you have the theory-gene, you will find the λ-calculus pretty daunting.

Bad news: the second most daunting model is Turing machines. They have a beautiful motivation and are very natural in a way, but when it comes to technical details they are a nightmare.

Alas, there is no choice: for complexity theory there is no way around Turing machines.

[†] Arithmetic functions defined by recursion over a *single* variable. To get full computability one needs recursion over any number of variables as in Herbrand-Gödel.