Mauro Brunato
DISI — University of Trento
Monday September 14, 2026

Computability and Computational Complexity

1. A Short History of Computability

Leibniz's Dream

Attempts at formalization of human reasoning in the hope of solving problems and disputes in a fair and unambiguous way (from a Western perspective — sorry).

- "Classical world" — Aristotle: logic, syllogisms; Euclid: postulates and theorems
- "Middle Ages" — Ibn Sina (Avicenna), Albertus Magnus, William of Ockham (Occam)...
- Gottfried Leibniz (1646-1716)

"Calculemus!"

- *Characteristica universalis*: universal formal language
- *Calculus Ratiocinator*: logical calculation framework

"If two friends should come to a disagreement, it would suffice for them to take their pencils in their hands and to sit down at the abacus, and say to each other — Let us calculate!"

- Gottlob Frege (1848–1925): *Grundgesetze der Arithmetik*
  - Fully formalized proof system. Every known mathematical theorem and proof could be expressed in his formalism!

Leibniz's Dream

What is a proof system?

- To prove a statement, however complex:
  - Start by asserting simple and obvious facts;
  - Break the proof into a sequence of steps, each based on previous ones (chain of reasoning).
  - Each step must be so simple and obvious that everyone would agree.
  - The final step asserts the proposed statement.
- At this point, everybody is "forced" to accept the statement!

Euclid's *Elements* (300 BCE) is an early example of such system.

Leibniz's Dream

More formally, a "proof system" is composed of:

- **Axioms** — A list of basic facts so simple and self-evident that everyone agrees on them.
  - Examples: "all right angles are equal", "the empty set exists"...
- **Theorems** — A dynamic list of known true facts.
  - In the beginning, the list of theorems is populated with just the axioms.
- **Deduction rules** — Combine known true facts to generate more true facts to be added to the theorem list.
  - Again, these rules must be simple, self-evident and unambiguous.
  - Example: *Modus Ponens* — the Master Tool of deduction
    if fact A is known to be true **and** fact $(A \Rightarrow B)$ is known to be true,
    then fact B is true and can be added to the list of known facts.

Leibniz's Dream

Frege's system was all of this:

- It began with a list of axioms and deduction rules about sets and numbers (*Grundgesetze*, basic laws).
- It painstakingly proceeded to derive more and more theorems about sets and numbers.

All of mathematics could, in principle, be derived within the system, e.g.:

"Julius Caesar is not a number" (discussed in the books).

Leibniz's dream is fulfilled!
. . . Or is it?

Leibniz's Dream at Risk!

Frege's axioms are very powerful: they can generate all of mathematics...

...and then something bad.

- 1903: Vol. 2 of *Grundgesetze der Arithmetik* is being printed at Frege's expenses.
- Bertrand Russell (1872-1970) writes to Frege: contradictions arise from his system.
- Frege takes the blow as a true gentleman, but he cannot hide his disappointment:

From Appendix A of *Grundgesetze der Arithmetik*, Vol. 2 (transl. from German)

"Hardly anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished. This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion."

Leibniz's Dream at Risk!

The problem is Frege's "Unrestricted comprehension" axiom, that he used to define sets based on the properties of their elements.

Definition (Frege's "Unrestricted comprehension" principle)

For every well-defined property $P(\cdot)$, there is the set

$$\{x \mid P(x)\}$$

of all and only the objects having that property.

Examples

- If $P(x) \equiv$ "x is green", then $G = \{x \mid P(x)\}$ is the set of everything that is green.
- Let $P(x) \equiv$ "x is a set". Then $S = \{x \mid P(x)\}$ is the set of all sets.

Note that both G and S are sets, therefore they are elements of S: $G \in S$, $S \in S$.
On the other hand, neither G nor S are green: $G \notin G$, $S \notin G$.

Leibniz's Dream at Risk!

Theorem (Russell's Paradox)

Every set theory that contains an unrestricted comprehension principle leads to a contradiction.

Proof.

If x is a set, $x \notin x$ ("x does not belong to itself") is a well-defined property. By the unrestricted comprehension principle, the set of all and only sets that do not belong to themselves exists:

$$R = \{x \mid x \notin x\}.$$

This introduces a contradiction, because $R \in R \Leftrightarrow R \notin R$. □

Popularized as the "barber paradox": in a village, the barber shaves all and only men who don't shave themselves. Does the barber shave himself?

Salvaging Leibniz's Dream

Ruling out unrestricted comprehension:

- Russell and Whitehead, *Principia Mathematica* (1910–1913) — from now on: "PM"
- Zermelo and Fraenkel set theory (1908–1922) — from now on: "ZF"

A property $P(x)$ can only be used to "filter" an already defined set. The notation

$$\{x \mid P(x)\} \cancel{\{x \mid P(x)\}}$$

is now illegal! We can only write

$$\{x \in A \mid P(x)\}$$

where A must be already defined (and must be in the domain of P).

The problem is solved!
. . . Or is it?

Hilbert's Dream

Even before the discovery of Russell's paradox, mathematicians were concerned about the germ of *inconsistency* (being able to prove conflicting statements).
David Hilbert (1862–1943) presented an influential list of 23 unsolved problems at the International Congress of Mathematicians in Paris in 1900.

Problem #2

Prove that the axioms of arithmetic are consistent.

- Both PM and ZF prevent self-referential definitions...
- ...and therefore avoid paradoxical objects such as Russell's set R...
- ...but more subtle inconsistencies might lurk inside them.

Hilbert's Shattered Dream

Indeed, the capability of formalizing integer arithmetic (possessed by both PM and ZF) is enough to enable self-referentiality *in a new way*.

- Kurt Gödel (1906–1978) — *Incompleteness theorems* (1931):
  - Reduction of logic to arithmetic: define a way to encode all of PM's logical statements as integer numbers and logical deduction rules as arithmetic calculations.
    - ⋆ This way, PM itself becomes able to describe its own statements, theorems and proofs.
  - Show a way to construct a number G corresponding to the statement "The statement represented by number G cannot be proved."
    - ⋆ If the statement is true, then by its own definition it has no proof:
      $\Rightarrow$ PM is *incomplete*, because there are true statements that it cannot prove.
    - ⋆ If the statement is false, then the converse is true, meaning that the statement can be proved, hence by definition it is true:
      $\Rightarrow$ PM is *inconsistent*, because it can prove a false statement.
- What's worse? Inconsistency is much worse, we would prefer PM to be incomplete...
- ...however, by Gödel's *second incompleteness theorem*, we cannot know which is the case.

Hilbert's Shattered Dream

OK, so PM (and ZF alike) cannot guarantee their own consistency, and, even if they are consistent, Gödel constructed an unprovable statement, so at the very least they are incomplete.
Moreover, Gödel's theorem implies that there's no way to make them complete.

So what?

- After all, Gödel's statement is extremely technical, and has been built on purpose.
  - In mathematical terms, it is a *pathological* example, unlikely to pop up in a real-world argument!
- Moreover, it isn't clear whether it's true or false!
- . . . Maybe (just maybe) all statements with some relevance can be proved true or false. . .

Problem ~~solved~~ averted?
Well. . .

Hilbert's Shattered Dream

Definition (Hilbert's Entscheidungsproblem, 1928)

Related to Problem #2 in Hilbert's 1900 list:
Is there an automated, unambiguous procedure to decide whether a given statement is valid using the rules of logic?

- Alonzo Church (1903–1995) and Alan Turing (1912–1954) independently answered


"**Nope!**"


  in 1936, using two different formalisms.
- In the first part of this course we will work on Turing's result.

A fundamental difference

We will see that Turing, like Gödel, shows that some statement is unprovable. However, unlike Gödel's, Turing's example isn't pathological and has an actual impact on Computer Science.

Will it halt?

$n \leftarrow 1$ ▷ *Start from the first odd number*
loop
  $sum \leftarrow 0$ ▷ *Sum all divisors of n*
  for $i \leftarrow 1 \dots n-1$ **do**
    if $n \equiv 0 \pmod{i}$ **then** $sum \leftarrow sum + i$
  if $sum = n$ **then halt** ▷ *If n is perfect, halt*
  $n \leftarrow n + 2$ ▷ *Else move to next odd number*

Definition

A natural number is *perfect* if it is equal to the sum of its divisors.
E.g., $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14, \dots$

- All perfect numbers we know are even: Are there odd ones too? We don't know.
- The code goes on until it finds an odd perfect number.
- Hence, we don't know if the code is ever going to halt.

Will it halt?

Recipe to create code that we don't know whether it will halt or not

1. Take an unproven conjecture in the form "All x are y".
   - See previous slide: "All perfect numbers are even."
   - Goldbach: "All even numbers are sum of two primes".
   - Riemann: "All nontrivial roots of $\zeta(x)$ have real part $1/2$".
2. Run a program that systematically looks for a counterexample.
3. ...
4. Profit!

Let's Turn the Tables.

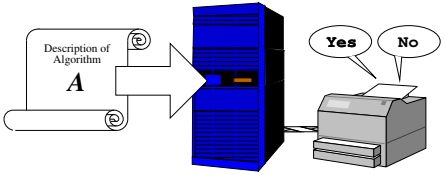
- Maybe we can find a general procedure that, given an algorithm A, tells us if A will eventually halt or not.
- Running ("simulating") A to check if it halts does not provide the answer:
  - if A halts we will know for sure that it does,
  - but if it doesn't halt we will never know!
- However, there might be some static analysis technique that looks at the algorithm and tells us if it will halt or not.

The Halting Problem

- Suppose that there is a function HALT(A) that receives an algorithm A as input and always returns a correct answer:

$$\text{HALT}(A) = \begin{cases} \text{true} & \text{if } A \text{ will halt} \\ \text{false} & \text{if } A \text{ will not halt.} \end{cases}$$

- In other words, suppose that we have a "machine", however made:



- We will prove that it cannot exist (i.e., it's going to give the wrong answer for some input.)

A contradiction?

Idea

Create a program R that invokes HALT asking it "What am I going to do?", then it does the opposite:

```
procedure R
    if HALT(R) then          ▷ If the HALT function predicts I am going to halt...
        loop                 ▷ ... then run forever.
    else                     ▷ If HALT predicts that I am running forever...
        halt                 ▷ ... then halt.
```

- This algorithm presents many challenges: it is self-referential, it isn't written in a well-defined programming language.
- In the next few lectures, we will learn how to make this argument solid and convincing.